PROJECTS ■ THEORY ■ APPLICATIONS ■ CIRCUITS ■ TECHNOLOGY

NUTS AND VOLTS

www.nutsvolts.com
July 2014

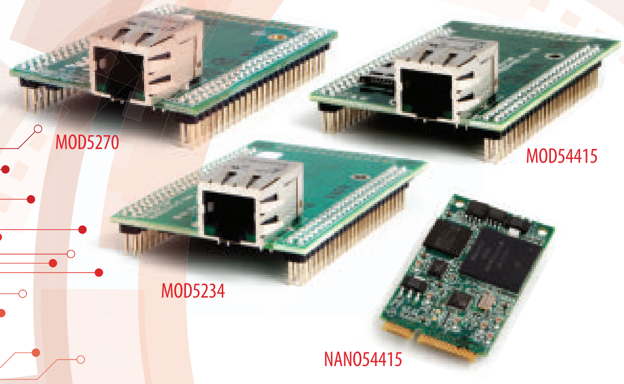
EVERYTHING FOR ELECTRONICS

Build The CAT AWAY

Electronic Cat Shooer

- Adafruit 32x32 RGB Matrix Panel
- PICAXE Download Cable

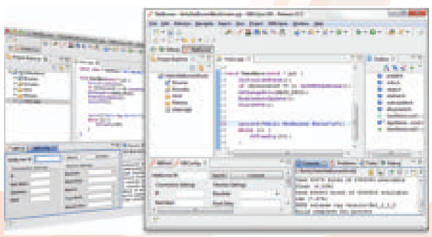
Ethernet Core Modules with
High-Performance Connectivity Options



- **MOD5270**
  147.5 MHz processor with 512KB Flash & 8MB RAM · 47 GPIO
  3 UARTs · I^2C · SPI
- **MOD5234**
  147.5 MHz processor with 2MB flash & 8MB RAM · 49 GPIO · 3 UARTs
  I^2C · SPI · CAN · eTPU (for I/O handling, serial communications, motor/timing/engine control applications)
- **MOD54415**
  250 MHz processor with 32MB flash & 64MB RAM · 42 GPIO · 8 UARTs
  5 I^2C · 3 SPI · 2 CAN · SSI · 8 ADC · 2 DAC · 8 PWM · 1-Wire® interface
- **NANO54415**
  250 MHz processor with 8MB flash & 64MB RAM · 30 GPIO · 8 UARTs
  4 I^2C · 3 SPI · 2 CAN · SSI · 6 ADC · 2 DAC · 8 PWM · 1-Wire® interface

Add Ethernet connectivity to an existing product, or use it as your product's core processor

The goal: Control, configure, or monitor a device using Ethernet



The method: Create and deploy applications from your Mac or Windows PC. Get hands-on familiarity with the NetBurner platform by studying, building, and modifying source code examples.

The result: Access device from the Internet or a local area network (LAN)

The NetBurner Ethernet Core Module is a device containing everything needed for design engineers to add network control and to monitor a company's communications assets. For a very low price point, this module solves the problem of network-enabling devices with 10/100 Ethernet, including those requiring digital, analog and serial control.

MOD5270-100IR........$69 (qty. 100)	NNDK-MOD5270LC-KIT..............$99
MOD5234-100IR........$99 (qty. 100)	NNDK-MOD5234LC-KIT............$249
MOD54415-100IR.....$89 (qty. 100)	NNDK-MOD54415LC-KIT..........$129
NANO54415-200IR...$69 (qty. 100)	NNDK-NANO54415-KIT...............$99

NetBurner Development Kits are available to customize any aspect of operation including web pages, data filtering, or custom network applications. The kits include all the hardware and software you need to build your embedded application.

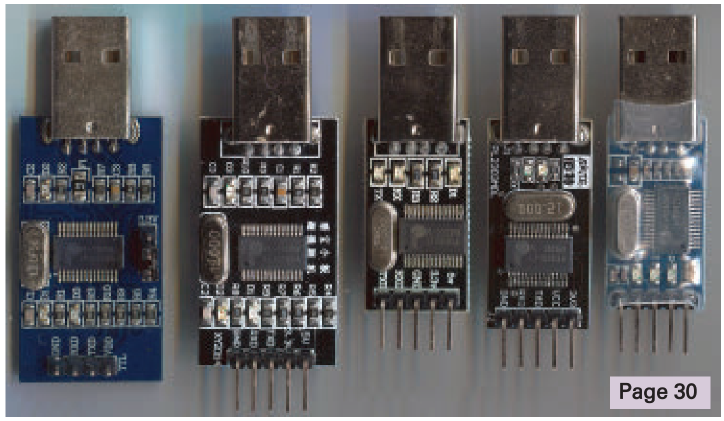
- **For additional information please visit**
  http://www.netburner.com/kits

NetBurner
Networking in One Day!

Information and Sales | sales@netburner.com
Web | www.netburner.com
Telephone | 1-800-695-6828

freescale™
Alliance Member

July 2014


Page 30

Columns

Departments

Nuts & Volts (ISSN 1528-9885/CDN Pub Agree #40702530) is published monthly for $26.95 per year by T & L Publications, Inc., 430 Princeland Court, Corona, CA 92879. PERIODICALS POSTAGE PAID AT CORONA, CA AND AT ADDITIONAL MAILING OFFICES. POSTMASTER: Send address changes to **Nuts & Volts, P.O. Box 15277, North Hollywood, CA 91615** or Station A, P.O. Box 54, Windsor ON N9A 6J5; cpcreturns@nutsvolts.com.

DEVELOPING PERSPECTIVES

Getting Away From the Bench

It's summer time and — especially if you've been cooped up all winter — time for outdoor activities. Because electronics experimentation is traditionally conducted indoors on a benchtop, you may put your hobby aside until after summer vacation. However, it needn't be the case. I'm not suggesting that you duplicate Benjamin Franklin's experimentation with kites and lightning, but there's plenty to do outdoors. For example, if you're into sports and training, there are any number of circuits you can build and then perfect outside — from timers for track and force sensing plates for impact sports, to radar guns for baseball.

If you're into drones and flying, then there are dozens of drone kits and hundreds of model airplanes and helicopters that you can build and modify — from adding a GPS system to an onboard video camera. My current favorite in miniature cameras is the GoPro series, especially if you're within range of a Wi-Fi hot spot. The remote control unit is easily duplicated, allowing you to design a control system that fits your particular needs. For GPS, there are several sub-$100 units available from SparkFun and other sources.

Don't forget that water and electronics often mix well. Whether you're installing a wireless water alarm system for your basement or to detect a stopped-up storm gutter, there's plenty you can do to improve your home. Last summer, I installed a weather station — complete with the usual detectors. For a twist, I added a Geiger-Muller tube and alarm to detect and notify me of an increase in the ambient radiation level. So far, I haven't detected any peaks in radiation.

If you're into tanning — or avoiding skin cancer — then consider an ultraviolet level/timer. It'll tell you when to flip over or when it's time to reapply your sunblock, depending on your goals. For supplies, consider an inexpensive UV filter made from a pair of sunglasses and a UV detector made from an inexpensive photodetector. Alternatively, consider an analog UV detector breakout board from SparkFun ($12). You'll also need your favorite microcontroller to process the analog signal and provide a suitable alarm.

As a final suggestion, consider repurposing or processing the signals from one of the popular physiological monitors on the market. For example, there's the Polar heart rate monitor interface also available from SparkFun ($60). Together with a microcontroller, you can develop any number of training routines for yourself or others. You'll need to program your maximum heart rate for your age, target value, and other parameters that you can look up in any good training manual. You might consider teaming up with a personal trainer if your knowledge of physiology isn't up to your abilities in electronics.

Whatever your outdoor interests, there's certainly a place for electronics. Now, get out there and get moving. **NV**

NUTS AND VOLTS

EVERYTHING FOR ELECTRONICS

Published Monthly By
T & L Publications, Inc.
430 Princeland Ct.
Corona, CA 92879-1300
(951) 371-8497
FAX **(951) 371-3052**
Webstore orders only **1-800-783-4624**
www.nutsvolts.com

Subscriptions
Toll Free **1-877-525-2539**
Outside US **1-818-487-4545**
P.O. Box 15277
North Hollywood, CA 91615

FOUNDER
Jack Lemieux

PUBLISHER
Larry Lemieux
publisher@nutsvolts.com

ASSOCIATE PUBLISHER/ ADVERTISING SALES
Robin Lemieux
robin@nutsvolts.com

EDITOR
Bryan Bergeron
techedit-nutsvolts@yahoo.com

VP of Operations
Vern Graner
vern@nutsvolts.com

CONTRIBUTING EDITORS
Joe Pardue
John Gavlik
Martin Hebel
Fred Eady
Jon McPhalen
Jeff Eckert
Paul Verhage
Dave Prochnow
Theron Wierenga
Alan Parekh
Tommy Tyler

CIRCULATION DEPARTMENT
subscribe@nutsvolts.com

SHOW COORDINATOR
Audrey Lemieux

MARKETING COORDINATOR WEBSTORE
Brian Kirkpatrick
sales@nutsvolts.com

WEB CONTENT
Michael Kaudze
website@nutsvolts.com

ADMINISTRATIVE ASSISTANT
Debbie Stauffacher

PRODUCTION
Sean Lemieux

ADVANCED TECHNOLOGY

■ The solar wind downdraft tower, scheduled to begin operation in 2018.

Tower of Power

No, we're not talking about the classic soul band, so we won't be "Diggin' on James Brown." We're focusing on Solar Wind Energy, Inc., which is offering "a bold new approach to overcome the current limitations of conventional wind energy sources." The company has announced the acquisition of a 600 acre chunk of real estate in San Luis, AZ, for the first US installation of its patented Solar Wind Energy Tower (SWET). The SWET is what is generically referred to as a solar wind downdraft tower. In operation, a mist of water is injected into the hot dry air at the top of the tower. The air absorbs the water, and thereby becomes cooler and denser. As a result, it drops downward at speeds up to 50 mph and is diverted into wind tunnels at the bottom where turbines are projected to generate up to 1,250 MW per hour.

It may be bold, but the concept isn't entirely new; it's really a twist on the solar updraft tower first proposed in 1903 by Isidoro Cabanyes, a Spanish army colonel. This design sucks hot air up through the tower using the chimney effect and generates power in pretty much the same way. An experimental model was in operation south of Madrid from 1982 to 1989, when its guy wires corroded and it fell over. The 33 ft (10 m) dia chimney rose to a height of 640 ft (195 m) and used a collection area (greenhouse) that covered 110 acres. The maximum output came in at about 50 kW. Several others have been built or proposed.

The downdraft version requires less land, and it has the advantage of operating 24 hours a day. However, pumping water to the top eats up an estimated 50 percent of the turbine's output, so it isn't an unmixed blessing. The concept has been promoted for 15 years or so by Prof. Dan Zaslavsky of Iowa State University. He estimates that — depending on construction costs and other variables — an energy tower should be able to offer power at a cost of between one and four cents per kilowatt which puts the price below other alternatives, except for hydroelectric. How well the San Luis facility will operate won't be known until it (maybe) begins operation in 2018, but it all sounds good. In the meantime, you can see a video on the company website (**www.solarwindenergytower.com**) and even invest if you are so disposed. As of this writing, the stock is moving at $0.03 per share. ▲

Laser Deployment Imminent

The US Navy has been seriously interested in laser weapons since 2010, when it awarded $11 million to Kratos Defense & Security Solutions (**www.kratosdefense.com**) to develop the Laser Weapon System (a.k.a., LaWS). By the time you read this, the first prototype should be installed on the USS Ponce for a 12 month at-sea test. The Ponce — a half century old Austin-class amphibious transport dock — isn't the most glamorous ship in the Navy, but it has been used as a test bed for several new technologies, including service as a "lilypad" for MH-53 mine-clearing helicopters.

The Navy has made directed-energy weapons a major priority to counter "asymmetric threats," including unmanned and small aircraft, and small attack boats. In a 2011 demonstration, a destroyer-based laser made short work of multiple small boat threats, and in 2012 LaWS downed several unmanned aircraft (YouTube videos are accessible by searching "navy laws laser"). As noted by Rear Adm. Matthew Klunder, "Spending about $1 per shot of a directed-energy source that never runs out gives us an alternative to firing costly munitions at inexpensive threats."

The news release from the Office of Naval Research was light on specifics, but it was revealed that "using a video game-like controller, a sailor will be able to manage the laser's power to accomplish a range of effects against a threat, from disabling to complete destruction." Its power level has been estimated at between 10 kW and 50 kW with an effective range of about a mile. More info is available at **www.onr.navy.mil**. ▲

■ The Navy's LaWS laser weapon, soon to be tested in the Persian Gulf.

COMPUTERS and NETWORKING

"Build Your Own" Gaming Machine

The average PC user is probably not familiar with CyberPowerPC (**www.cyberpowerpc.com**), as the company's product line tends to gravitate toward high-end gaming machines. The Los Angeles based company's approach is to be both manufacturer and distributor, thereby offering customers high tech products at wholesale prices. If you are a present or aspiring gamer, you might want to check out CyberPowerPC's "build your own" desktop bundle available through (no kidding) Walmart. This give you the option of choosing among a range of cases, motherboards and processors, drives, video cards, and so forth to create a totally custom machine. The bottom of the line configuration runs only $465, but processing power comes from an AMD FX-4100 CPU which rates an average CPU benchmark of only 4045, according to **cpubenchmark.net**. You can go all the way up to an Intel Core i7-4960X Extreme Edition, however, which tests out with a 14164 benchmark. This option will run an extra $1237, so you'll need to be serious about being serious.

Of course, the power of the graphics card is crucial for gaming, and the el-cheapo version comes with a lowly CyberPower CPVC6100 NVIDIA GeForce GT 610 card. For an extra $794, you can substitute a GeForce GTX 780Ti — which the company claims is "the best gaming GPU on the planet." In all, you can choose from a list of 15 different motherboards and processors, 15 graphics cards, six hard drives, three optional cooling systems, and many other options. Don't worry if you're not mechanically inclined. The "build your own" phrase isn't really accurate. CyberPowerPC actually assembles the machine at the factory, loads the OS, and tests it before shipping it to you. ▲

■ Raidmax Viper gaming case, one of six options.

New Ruggedized Notebooks

We don't usually associate Dell, Inc. (**www.dell.com**), with military-grade equipment, but the company recently introduced the Latitude Rugged Extreme lineup, built for "the world's toughest jobs in the harshest conditions." The Latitude 12 notebook and Latitude 14 convertible notebook models are built to withstand dust, moisture, vibration, extreme temperatures, and drops from up to six feet. Plus, data is protected with sealed doors and compression gaskets. The cases are built with impact-resistant polymers and magnesium alloy, and the machines have been independently tested for compliance with MIL-STD-810G, IEC 60529 (IP-65), MIL-STD-461F, and ANSI/ISA.12.12.01 standards. Other interesting features include an outdoor readable display — even in harsh sunlight — a full HD webcam with privacy shutter, a backlit keyboard with an "instant stealth" mode, and an eight megapixel bottom camera with flash (Latitude 12 only). The Latitude 14 (starting at $3,499) offers 14 hr of battery life, and the Latitude 12 ($3,649 up) will run for 8.5 hr on a charge. ▲

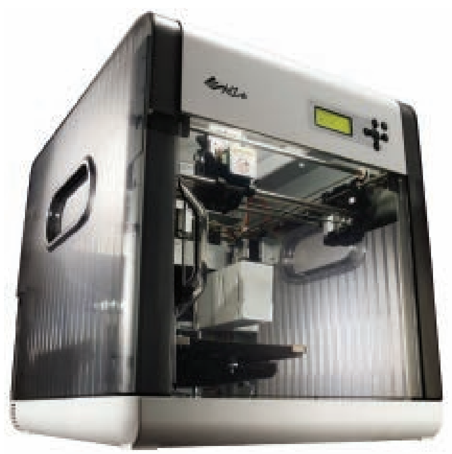
■ Dell's Latitude Rugged Extreme notebooks withstand dust, moisture, extreme temperatures, and drops from six feet.

COMPUTERS and NETWORKING Continued

Free Disposable Email

It's a common occurrence. While browsing online, you want to leave a comment, write a review, or otherwise contribute to a discussion, but the site requires you to sign up for an account and verify your email address first. That would be okay except that they often sell your address to a couple thousand spammers, and your inbox is soon filled with unsolicited garbage. Now, there is an easy solution: just go to **mailinator.com**. Mailinator provides free, anonymous, disposable email addresses. You don't have to sign up or use a password, and you can use any name you want. If you want to be suzycreamcheese@mailinator.com, go ahead. If anyone sends an email to that address, all you have to do it log on and search on that name to read the message. You don't even have to use **mailinator.com** as the domain; hundreds are available. The Mailinator folks won't reveal the full list, but they do suggest some from time to time. For example, @zippymail.info works, as does @reallymymail.com. One limitation is that Mailinator is receive-only, so you can't use it as an outgoing address. Another is that it is not at all private, so anyone who knows what address you used can log in and read your messages, so it's probably not a good idea to have anything important sent to you. On the positive side, all messages are automatically deleted after a few hours, so you don't have to worry about junk building up over time. ▲

CIRCUITS and DEVICES



■ The Da Vinci 1.0 3D printer breaks the $500 price barrier.

3D Printing Gets Cheaper

If you have been waiting for a 3D printer that costs less than $500, wait no longer. The Da Vinci 1.0 printer from XYZprinting (**us.xyzprinting.com**) is a plug-and-play machine that squeaks in at $499. The 2014 CES Editor's Choice Award winner uses both ABS and PLA filament materials which come in 12 colors. The Da Vinci uses fused filament fabrication (FFF) technology to print items in sizes up to 7.8 in (198 mm) in all three dimensions, and its fully enclosed design protects users from the high temperatures and keeps byproducts from being released. You don't necessarily have to know anything about programming to put it to use, as owners can log into the company website and choose downloadable 3D object files from thousands of designs. Filament cartridges run a fairly reasonable $28 each, so you can make a slew of key chains, cups, smartphone cases, and so on without breaking the bank. ▲

Another TV Streamer

It's a slow process, but streaming TV is making steady inroads against real time broadcast entities. There are already many paid programming sources — including Hulu, Netflix, and Amazon Instant — plus a huge number of very cheap and even free services (perhaps the oddest of which is Nowhere TV, **thenowhereman.com**). There are also several devices that can be used for streaming, including the various Roku models, Google's Chromecast, Apple TV, and PlayStation 3. To that list, we can now add the Amazon Fire TV. It's a $100 box that — like some of the others — also allows users to access video games and stream photos and music into an HDTV. According to Amazon, the box provides more than three times the processing power of its competitors, based on a quad-core Snapdragon 600 processor with an Adreno 320 graphics engine and 2 GB of memory. It also offers 1080p video, 7.1 Dolby Digital Plus surround sound, and a dual-antenna Wi-Fi connection. Perhaps the most interesting aspect is the voice search feature: You just press a button on the remote, speak the words "Green Acres," and soon you'll be face to face with Arnold Ziffel. The Amazon Fire is available, of course, at **www.amazon.com**. ▲

■ The Amazon Fire TV includes a voice-activated search.

INDUSTRY and the PROFESSION

HP Teams with Foxconn

The Taiwan-based Foxconn Technology Group (**www.foxconn.com**) is probably best known for manufacturing iPads and iPhones but, in fact, also cranks out BlackBerries, Kindles, PlayStations, and many other devices. As of May 1, Hewlett-Packard (**www.hp.com)** has joined the flock by way of a joint venture to create a new line of cloud-optimized servers specifically targeting service providers. Apparently, HP has taken notice of a report from International Data Corporation (IDC, **www.idc.com**) predicting that — as a result of a boom in cloud-based delivery services — there will be "hyperscale growth in servers used for hosting (15 to 20 percent compound annual growth rate from 2013 to 2018)."

According to Meg Whitman, HP president and CEO, "With the relentless demands for compute capabilities, customers and partners are rapidly moving to a new style of IT that requires focused, scalable, and high volume system designs. This partnership reflects business model innovation in our server business, where the high volume design and manufacturing expertise of Foxconn — combined with the compute and service leadership of HP — will enable us to deliver a game-changing offering in infrastructure economics."

The new server line will complement — not replace — HP's existing ProLiant servers, including Moonshot. Details on pricing and availability were not available. **NV**

Getting Fancy With LANC

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/index.php?/magazine/article/july2014_SpinZone**.

I think most of my regular readers know that my life is pretty evenly split between technology and entertainment — though a good bit of my technology work is for the entertainment industry. Having been inspired by the Robert Rodriguez book, *Rebel Without a Crew*, I bought my first serious camcorder (Sony VX-1000) and learned how to shoot and edit video. One of the first accessories I purchased was a zoom and focus controller that plugged into the LANC (pronounced LAN-see) port of the camera. Every time I looked at that little port I thought, "What else could I do with this?"

Some years after that purchase, I met a guy named Lou from Camera Turret Company. He wanted to build his own focus and zoom controller, but didn't have the requisite embedded design or programming skills. At the time, I had just transitioned from Parallax to EFX-TEK where I was developing a lot of products using the SX microcontroller, doing my programming in a mix of SX/B (BASIC) and SX assembly.

It took a few weeks, but we had a working prototype. Sadly, that product never went into production. Still, it did whet my appetite for LANC. This past January, Lou called and asked if it would be possible to do LANC with the Propeller. Of course!

We pulled it off with the SX, so we can easily do that with the Propeller. In fact, we can do a whole lot more than we accomplished the first time around.

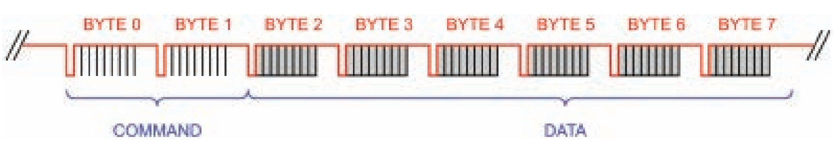
Under the Hood of LANC

LANC (Local Application Control Bus; a.k.a., Control-L) has been available for quite a long time; hence, is very simple. From the hardware point of view, LANC is an open-collector serial connection that runs at a very tame 9600 baud. The camcorder transmits an eight-byte packet to the controller 60 times per second (NTSC systems). **Figure 1** shows the make-up of the LANC packet. Here's the neat part: The first two bytes of the packet are always $FF, and the camcorder reads these back as it's transmitting them. If a connected controller wants to pass a command to the camcorder, it simply modifies these bytes.

But how? As I just stated, the connection between the camcorder and controller is open-collector. What this means — as in I^2C that we covered last time — is the serial connection is pulled high through a resistor; this sets the idle state and "1" bit state of the line. If either side wants to create a "0" bit, it pulls the line low. As in I^2C, neither side can drive the line high; that is always left to the pull-up. Yes, LANC is just serial data, and pretty slow at that. We've created software UARTs with the Propeller many times, so this one won't be a lot of work.

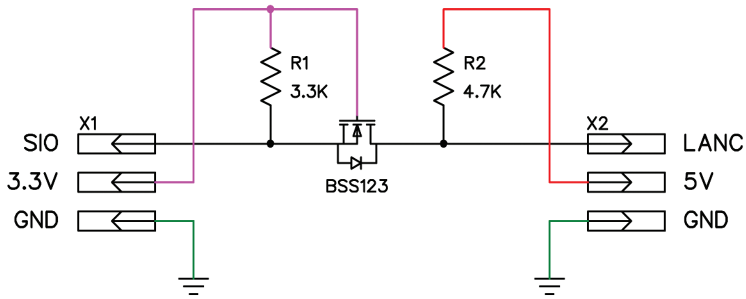
Getting Connected

Figure 2 shows a very simple circuit for connecting the Propeller (or any other 3.3V micro) to the LANC line. If it looks familiar, you may remember it from the Dynamixel project we did last fall. Yes, this is a very simple bi-directional 3.3V to 5V level shifter; this circuit is

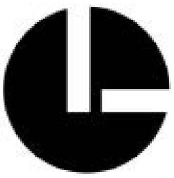


■ FIGURE 1. LANC packet.

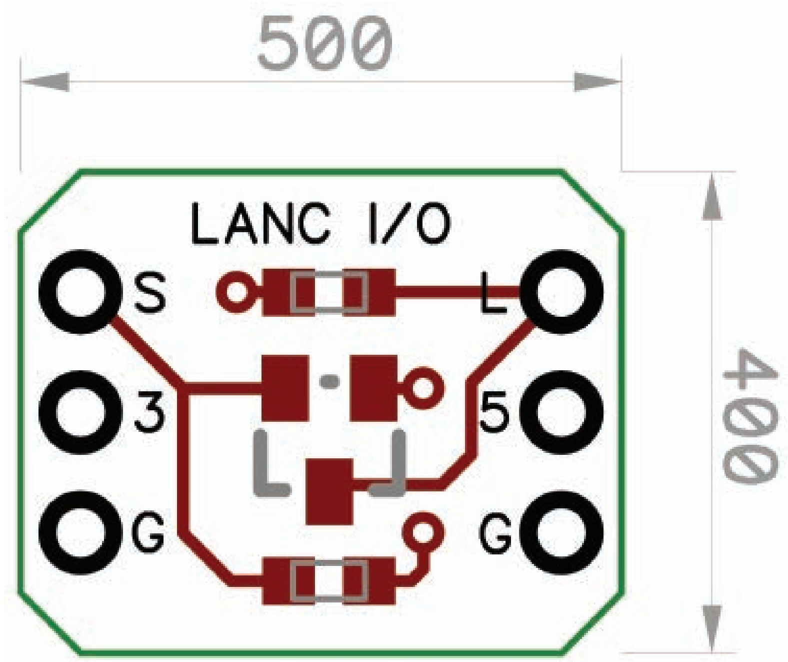
■ FIGURE 4. LANC plug.



■ FIGURE 2. LANC interface schematic.



■ FIGURE 5. LANC logo.



■ FIGURE 3. LANC interface PCB.

■ FIGURE 6. LANC to AV adapter.

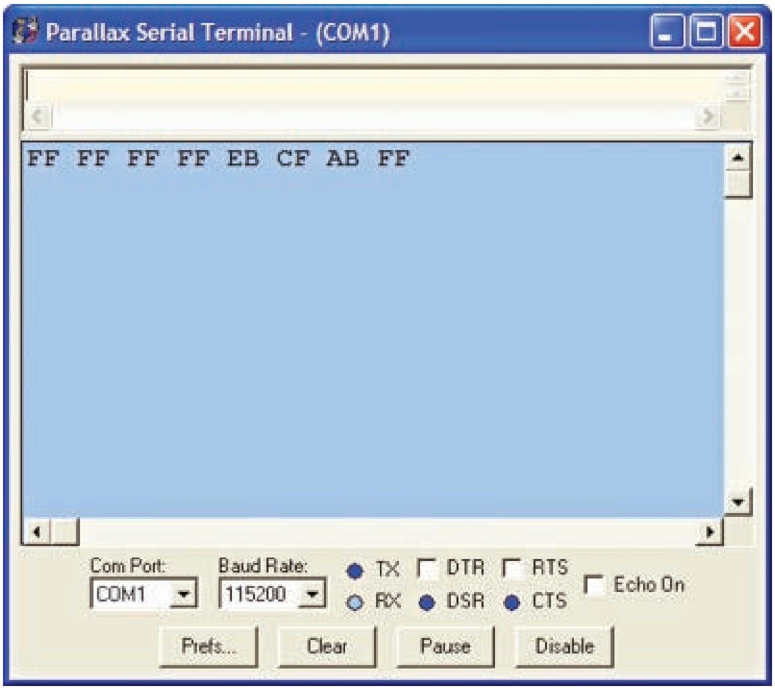
frequently used for serial and I²C buss connections.

I happen to have this circuit on one of my favorite Propeller controllers: the EFX-TEK HC-8+ (the SIO port). If you don't, I've also laid out a little breadboard-friendly module (**Figure 3**) that you can use for experimenting with LANC.

The typical LANC port on a camcorder is a 2.5 mm stereo jack. **Figure 4** shows the connections on the controller's stereo plug. I don't use the 5V from the camera; this voltage goes away when the camera goes into sleep mode while the pull-up is maintained. This allows a "sleeping" camera to be awakened by pulling the LANC line low for more than 140 ms. Using the circuit in **Figure 2**, we can do this via software control.

The LANC jack will be marked as such; usually with the logo shown in **Figure 5**. If you have a Sony camcorder that uses a 10-pin A/V jack, you'll need to get an adapter. I prefer to buy my camera gear from reputable companies like B&H Photo in New York; their part number for an adapter is #LIA3AV. If you would rather shop elsewhere, do a web search on "LANC AV adapter" and you'll be presented with a lot of choices. What you're looking for is shown in **Figure 6**.

From a basic software standpoint, LANC is very simple: The camcorder sends an eight-byte packet 60 times per second (every 16.7 ms). LANC bytes are eight bits, plus a start bit and three stop bits. Yes, three. Remember, LANC has been around for a long time and in the early days, micros were not nearly as fast as they are today. Those extra stop bits gave early micros enough time to process each byte. What this all boils down to is that we have about 6.7 ms between packets. As with



■ FIGURE 7. LANC input test.

MODBUS and similar protocols, we'll look for this idle period to set up for the start of the next packet/frame.

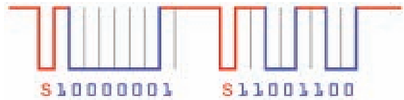
Let me show you how easy this is using a standard serial object:

```
pub main | c

  lanc.start(SIO, SIO, %1100,    9_600)
  term.start(RX1, TX1, %0000, 115_200)
  pause(10)

  term.tx(CLS)

  repeat
    repeat
      c := lanc.rxtime(2)
    until (c < 0)

    term.tx(HOME)

    repeat 8
      term.hex(lanc.rx, 2)
      term.tx(" ")
```

In this example (***jm_lanc_monitor_ez.spin***), we're using two copies of FullDuplexSerial: one for the LANC input; the other for displaying the LANC data on the PST terminal. The LANC serial is set up for half-duplex/open mode at 9600 baud.

After clearing the terminal, the program drops into a loop. At the top, we use the ***rxtime()*** method to wait for the idle space between packets. Here's how that works: ***rxtime()*** will wait up to *n* milliseconds for a character. If no character is available, it will return -1. By waiting for 2 ms and receiving a -1, we know that we've located the idle gap between packets. Another loop displays the next eight bytes in hex format. The output of the program is shown in **Figure 7**.

When you run this program, you'll see the last six bytes flipping around, though the first two — both $FF — will never change. These are the command bytes and are changed by an external controller. That will be the focus of our LANC object.



■ FIGURE 8. LANC command bytes.

Commanding LANC

LANC command sequences are two bytes, and these are expected at the beginning of the packet. The responsibility of the external controller is to wait for the command bytes and then modify them during transmission. Again, this is possible due to the open-collector nature of the LANC connection.

The camcorder will always output $FF, which is to say that the pull-up is keeping the LANC line high for each bit. This allows the controller to modify these bytes by pulling selective bits low.

Building a serial UART for 9600 baud is no trouble at all for the Propeller; in fact, it's easily done in Spin. There is a special consideration with sending command bytes for LANC: The camcorder generates the start bits for each byte. To be candid, Spin is probably fast enough to deal with this, too, but I chose to write my driver in PASM; it's easy enough to do, and we have better precision over timing.

Have a look at **Figure 8**. This diagram represents the LANC line when the command bytes $18 and $33 (record toggle) are being transmitted. The red line indicates control of the LANC serial line from the camcorder side; the blue line represents control of LANC serial from the controller side. Note that the controller can only pull the LANC line low; the pull-up in the camcorder keeps the line high for the idle state and one bit. The critical point is that the camcorder exerts the start bits for all bytes in the packet. That is to say that our serial output to the LANC device will be modified from

previous UARTs. Instead of creating the start bit, we'll wait on one, then skip passed it and modify any bits required for the command.

For those worried about synchronization, don't. Keep in mind that UARTs will sample around the middle of each bit period, so a small offset will do no harm. Okay then, let's jump in and build a full-featured LANC object.

With a fixed baud rate, the only thing the LANC ***start()*** method needs from us is a pin to use. This method will set up the mask and timing values for LANC serial, as well as timing requirements for finding the idle period, detecting the loss of the LANC stream and setting up a couple pointers:

```
pub start(sio)

  stop

  cogparms[0] := 1 << sio
  cogparms[1] := clkfreq / 9600
  cogparms[2] := cogparms[1] * 15 / 10
  cogparms[3] := (clkfreq / 1000) << 1
  cogparms[4] := clkfreq >> 2
  cogparms[5] := @framecount
  cogparms[6] := @buffer

  usercmd := @cogparms

  cog := cognew(@lancio, @usercmd) + 1

  return cog
```

Of late, I have been using a simple array to hold values that will be moved into the cog. This is easier than trying to create similar sounding yet unique variables — especially since they won't be used by the high level code. We go through this trouble so that the PASM code can be used by other languages.

The first of the parameters (*cogparms[0]*) is the mask for the LANC serial pin. Next up are the timing values (in system ticks) for 1.0 and 1.5 bits. The next parameter is the number of ticks in 2 ms; this is used to detect the idle period between packets.

What follows is the number of ticks in 250 ms; this is used to detect a loss of signal, as well as the time to exert for waking a camera that has gone into power-down (sleep) mode. Finally, we provide pointers to hub variables that hold the received frame count and the LANC serial buffer.

The hub address (@) of the parameters array is moved into *usercmd*, and the address of *usercmd* is passed to the cog in the ***par*** register via the **cognew** instruction.

On the PASM side, we have a block of variables that looks like this:

```
siomask        res     1
bit1x0         res     1
bit1x5         res     1
idletix        res     1
lostix         res     1
fcaddr         res     1
bufaddr        res     1
```

These variables must match the order that we set up on the Spin side, as we will simply copy the values from the hub into the cog with a loop:

```
lancio       rdlong      r1, par
             movd        :read, #siomask
             mov         r2, #7
:read        rdlong       0-0, r1
             add         r1, #4
             add         :read, INC_DEST
             djnz        r2, #:read
```

We begin by reading the content of ***par*** into cog variable *r1*. The content of *r1* will be the hub address of the variable *usercmd*; the content of *usercmd* is the address of our parameters array. This may, at first, seem a little convoluted. The cog needs to know the hub address for a command that we will send later. Since we can only pass one value in ***par***, we pass the address of our command variable; then for setup, we use the command variable to transport the address of parameters required by the cog.

The next instruction, **movd**, moves the cog address (#) of *siomask* to the destination register of the instruction marked ***:read***. We have seven parameters, so this value is moved into *r2*.

The working loop starts at ***:read***. This will read from the hub address in *r1* to the cog address that is presently in the destination field. In case you haven't seen it before, the 0-0 in the **rdlong** instruction designates a field that will be modified by the code. This has initially been set to the cog address of *siomask*.

After reading a value from the hub to the cog, the value in *r1* is incremented by four to point to the next long in the hub. A constant value called **INC_DEST** is added to ***:read*** to advance the destination field to the next cog variable. After seven iterations through the loop, we have successfully transferred all the parameters required by the cog. We did most of the work in the **start()** method, so there's not much left to do on the PASM side before dropping into the LANC code:

```
             andn     outa, siomask
             andn     dira, siomask

             mov      ctra, FREE_RUN
             mov      frqa, #1
```

```
              mov      phsa, #0

              mov      lancframes, #0


clear_cmd     mov      r1, #0
                    wrlong   r1, par
```

We use **andn** to write a 0 into the output and direction bits of the LANC pin; this causes the pin to be in input state and preps the output bit for use later.

Next, we set up ***ctra*** to free run; this will be used to detect the loss of LANC data. Lou's project, for example, uses a wireless connection between a base station and several remote cameras; by detecting the loss of the LANC stream, his system can tell if a camera has gone off-line or was disconnected.

Finally, we clear the frames received counter and the value presently sitting in the command variable (*usercmd*, which is holding the hub address of *cogparms[0])* — this lets the foreground know that we're ready.

The main control loop is fairly trim: We check to see if there is a command from the application and if so, extract the command bytes and count. If there is no command, we'll simply wait on the next LANC packet and move it into the hub:

```
lanc_loop      rdlong   cmd, par
               mov      cmdcount, cmd
               and      cmdcount, #$FF
               tjz      cmdcount, #rx_lanc8

               cmp      cmdcount, #$FF   wc, wz
        if_e   jmp      #wake

               mov      cmdb1, cmd
               shr      cmdb1, #8
               and      cmdb1, #$FF

               mov      cmdb2, cmd
               shr      cmdb2, #16
               and      cmdb2, #$FF
```

As we will see in just a moment, the command bytes and transmission count are sent in a packed long. This is read from the hub (the address of *usercmd* is in ***par***) and the count is extracted. If the count is zero, there is no command, and we can receive the next packet and move it to the hub. Let's do that.

As you now know, we synchronize with the LANC stream by looking for the idle period between packets. That is handled with this code:

```
wait_idle         neg      r1, cnt
:loop             test     siomask, ina   wc
    if_nc         jmp      #wait_idle
                  mov      r2, r1
                  adds     r2, cnt
                  cmps     r2, idletix   wc, wz
    if_b          jmp      #:loop
wait_idle_ret     ret
```

As I've stated many times, one of my favorite aspects of the Propeller is the ability to do precise timing by using the ***cnt*** register, and that's what we're doing here. At the top of the loop, we move the negative value of the ***cnt*** register into *r1*; this is the starting point of our timing. Then, we drop into a loop that looks at the LANC serial line, moving the state of the line into the Carry bit. If the line was low (**nc**), we start all over because we've sampled in the middle of a zero bit; this means we're not in the idle period.

If the LANC serial line is high, we copy the starting value into *r2* and add this to the current content of the ***cnt*** register. At this point, *r2* holds the time (in system ticks) between when we (re)started and now. This is compared with the value in *idletix*. Once the value in *r2* exceeds the value in *idletix*, we know that we've found the idle period, and can exit and receive the packet:

```
receive          mov      rxwork, #0
                 mov      rxbits, #8
:loop            mov      rxtimer, cnt
                 test     siomask, ina   wc
    if_nc        jmp      #rx_byte

                 cmp      lostix, phsa   wc, wz
    if_a         jmp      #:loop

                 mov      losflag, #1
                 jmp      #receive_ret
```

BILL OF MATERIALS

Q1	BSS123	Mouser 512-BSS123
R1	3.3K	Mouser 71-CRCW0603-3.3K-E3
R2	4.7K	Mouser 71-CRCW0603-4.7K-E3
X1,X2	0.1x3 M	Mouser 2340-6111TG
PCB	BayArea Circuits	

RESOURCES

Jon "JonnyMac" McPhalen
jon@jonmcphalen.com

Parallax, Inc.
www.parallax.com
Propeller chips and programming tools

www.boehmel.de/lanc.htm
Excellent LANC reference

```
rx_byte          add      rxtimer, bit1x5
:loop            waitcnt rxtimer, bit1x0
                 test     siomask, ina  wc
                 shr      rxwork, #1
                 muxc     rxwork, #%1000_0000
                 djnz     rxbits, #:loop
                 waitcnt rxtimer, #0
                 xor      rxwork, #$FF
                 mov      phsa, #0
                 mov      losflag, #0
receive_ret      ret
```

Long time readers may recognize this code as it is liberated from my DMX receiver object. It's a simple eight-bit receive UART that also checks the state of a free-running timer to make sure we have not lost the intended data stream.

At the top, we clear the workspace (*rxwork*), set up for eight bits, and then grab the current value of the ***cnt*** register. Then, we sample the serial line looking for the start bit. If there is no start bit, we look at the free-running value in ***ctra***. If this exceeds the value of *lostix*, we abort the receive process.

Losing signal is rare. Most of the time, we will get a start bit and jump down to **rx_byte**. At this point, we add the timing for 1.5 bits to *rxtimer* and drop into the receive loop. At the top of this loop is a **waitcnt** that will allow us to sample in the middle of each bit. The serial line is sampled and moved into the Carry bit.

As serial data arrives LSBFIRST, we right-shift the workspace and then move the Carry bit into rxwork.bit7 using **muxc**.

After allowing the final bit period to expire, we invert the received bits with **xor**, then restart the loss-of-signal timer and clear the loss-of-signal flag.

You may be wondering why we inverted the received bits. The interface shown in **Figure 2** is non-inverting. If you look at the standard Sony circuit implementation, it inverts the bits between the LANC device and the processor. Hence, the command bytes are documented for the inversion. We invert bits in software so that we can use the command bytes from the Sony documentation and other sources on the Internet.

If you change the circuit to an inverting hardware interface, you'll need to remove the inversion (this is marked in the fully commented code at the article link). Once we have a byte, it is written to the hub and we will continue that process until the LANC packet is received. From there, we go back to the top and look for a command. Let's go back to the Spin interface code and see how the command value is built:

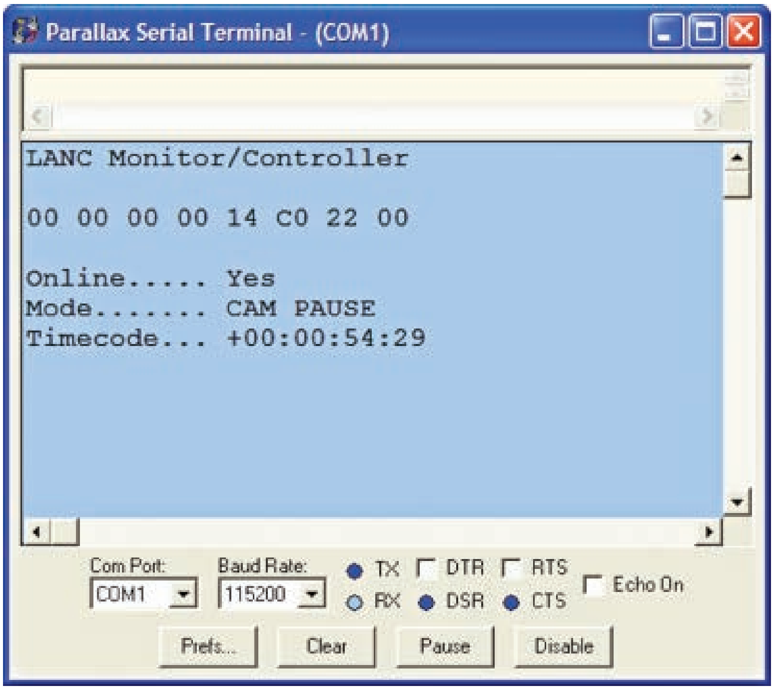
```
pub command(lanc1, lanc2) | uc

  if (framecount == 0)
    return false

  ifnot (busy)
    uc.byte[0] := REPS
    uc.byte[1] := lanc1
    uc.byte[2] := lanc2
    uc.byte[3] := 0

    usercmd := uc
    return true

  else
    if (lanc1 <> usercmd.byte[1])
      return false
    elseif (lanc2 <> usercmd.byte[2])
      return false
    else
      usercmd.byte[0] := REPS
      return true
```

The ***command()*** method requires two bytes. The first thing we look for is an active LANC signal. We know this is true when the received frame counter is not zero.

If we're not already busy sending another command, we build a command long using the temporary variable *uc*. It's important to build the command in a temporary variable so that it is complete before we allow the PASM code to read it. The constant called **REPS** is the repetitions for each command — typically, three or four (I use four). With the packed command built, we can move it into *usercmd* so the PASM cog can act on it.

If the PASM cog is already busy with another command, we check to see if it matches the last. If that is the case, we update the **REPS** value. Why would we do this? If you build a camera controller and want to press and hold a command key (e.g., Zoom In), this will keep that command going to the camcorder until you release the button. If the new command does not match the current, it is ignored and the method will return ***false***. Okay, then. Let's look at the LANC transmit UART. This is just a little different from UARTs we've built in the past:

```
transmit       mov      txbits, #8
               xor      txwork, #$FF
:loop          mov      txtimer, cnt
               test     siomask, ina  wc
    if_nc      jmp      #txbyte

               cmp      lostix, phsa  wc, wz
    if_a       jmp      #:loop

               mov      losflag, #1
               jmp      #transmit_ret
```



■ FIGURE 9. LANC control demo.

```
txbyte          add       txtimer, bit1x0
                waitcnt   txtimer, bit1x0

tx_bit          test      txwork, #1   wc
                muxnc     dira, siomask
                shr       txwork, #1
                waitcnt   txtimer, bit1x0
                djnz      txbits, #tx_bit
                andn      dira, siomask
                mov       phsa, #0
                mov       losflag, #0
transmit_ret   ret
```

As I've stated, the camcorder will create the start bits for all bytes in the packet. That means our transmit code has to use an external start bit instead of creating it. We start by setting up for eight bits and inverting the byte to match our hardware interface. Then, we drop into a loop that is identical to what we did on the receive side; while waiting for a start bit, we check the loss-of-signal timer.

Once the start bit is detected, we jump to ***tx_byte*** where we add the value of one bit period to the timer value. In this case, we want to match with the starting edge of the bit, not the middle as in receive. At ***tx_bit***, we copy a bit from the command byte into the Carry flag, and then use **muxnc** to move it to the output using the direction bit of the pin.

Remember, we've already set up the output bit with a zero. What this code does is make the LANC pin an output whenever we have a zero in a command byte. This will put the LANC line low which is seen by the camcorder (refer to **Figure 8** blue lines).

After sending a command byte, we clear the loss-of-signal timer and flag. When both bytes have been transmitted, we drop into a loop that will receive the next six bytes of the stream and move them to the hub buffer (see full listing).

That's the hard work of the PASM code. There is one small bit that is used in special cases. If the camcorder is put into or goes into power-down/sleep mode, we can wake it by pulling the LANC line low for at least 140 milliseconds. This is not a serial signal; it's a brute force thing. Many LANC devices (including my first controller) have a normally-open pushbutton between the LANC line and ground; this button is used to wake a sleeping camera. The interface we're using allows us to do this with software. The wake command puts $FF into the repetitions byte. When this is detected by the command processor, it jumps to this short bit of code:

```
wake      andn      outa, siomask
          or        dira, siomask
          mov       r1, lostix
          add       r1, cnt
          waitcnt   r1, #0
          andn      dira, siomask
          mov       phsa, #0
          jmp       #clear_cmd
```

We start by making the LANC serial line an output and low, then moving the value of *lostix* (250 ms) into *r1*. We add in the value in ***cnt*** and use **waitcnt** for the delay. When the delay is finished, we release the LANC line to the pull-up, then clear the loss-of-signal timer as the camcorder should now be awake and sending a LANC stream.

LANC in Action

In the downloads package at the article link, you'll find a little demo program that will decode the LANC stream and display the status on a terminal as shown in **Figure 9** (note that the bytes are inverted using the new interface to match published LANC codes). This program also accepts keyboard commands which allow us to explore camcorder control from our applications.

I've included the basics: power control; zoom and focus control; and record start and stop. Commands for these came from Sony documentation and from an excellent website that you'll find in the **Resources** block.

In the main loop of the program, we decode key

input to the terminal like this:

```
repeat
  c := ucase(term.rxcheck)
  if (c => 0)
    case c
      "P": { power }
        if (lanc.available)
          lanc.command($18, $5E)
        else
          lanc.wake_up
        term.rxflush

      "R": { toggle record }
        lanc.command(18, $33)
        term.rxflush

      "A": { toggle auto focus }
        lanc.command($28, $41)
        term.rxflush

      "N": { focus near }
        lanc.command($28, $47)

      "F": { focus far }
        lanc.command($28, $45)

      "T": { zoom tele - fast }
        lanc.command($28, $0E)

      "W": { zoom wide - fast }
        lanc.command($28, $1E)
```

As you can see, we flush the terminal keyboard buffer when implementing a command that toggles an aspect in the camcorder. This prevents accidents from holding the key down too long.

Note, too, the special case with the power command; if no LANC stream is available, we can assume the camera is in sleep mode. In this case, we use the ***wake_up()*** method. In practical applications, you may want to re-check the status of the LANC line after a wake-up to ensure the camera is actually connected.

I'll leave you to explore the rest of the program; it's pretty easy. What you'll find is there is a lot of data in the LANC stream that can be displayed — time-code, for example, is an important element when shooting video.

As a final note, I want to caution readers about purchasing LANC documentation from any source other than Sony. While working on Lou's last project, I came across an engineering company that offered a PDF on LANC and its implementation; the price for the PDF was $20. I figured — coming from an engineering company — it would be worth the expense.

Sadly, it wasn't even worth the paper I paid to print it on. The example PIC circuit was 20 years old; the code was badly formatted and not well described or commented. Save your cash (or use it to extend your subscription to *Nuts & Volts*). Refer to the link in the **Resources** block for top-notch information on LANC.

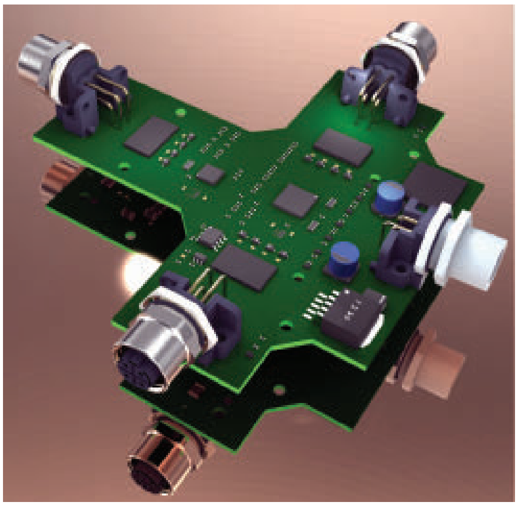
Until next time, then, keep spinning and winning with the Propeller!

Oh! Hey! I almost forgot! If you're going to be at DEF CON 22 next month, please find me and say "Hello!" I'll be spending most of my time in the Hardware Hacking Village. See you in Vegas! **NV**

NEW PRODUCTS

- HARDWARE
- SOFTWARE
- GADGETS
- TOOLS

FREE 3D DATA FOR EAGLE PCB LAYOUT SOFTWARE



The printed circuit board (PCB) manufacturer Beta LAYOUT Ltd is now introducing a virtual 3D EAGLE viewer in its PCB-POOL®. The unique tool is called "brd-to-3D" and it works with a proprietary library.

By uploading the EAGLE layout file (with more formats coming soon), the customer receives a "free" comprehensive 3D package consisting of:

- Photo-realistic images of the PCB and SMD stencil.
- A STEP file of the virtually assembled PCB.
- A PDF formatted 3D view that rotates freely allowing all angles of viewing.

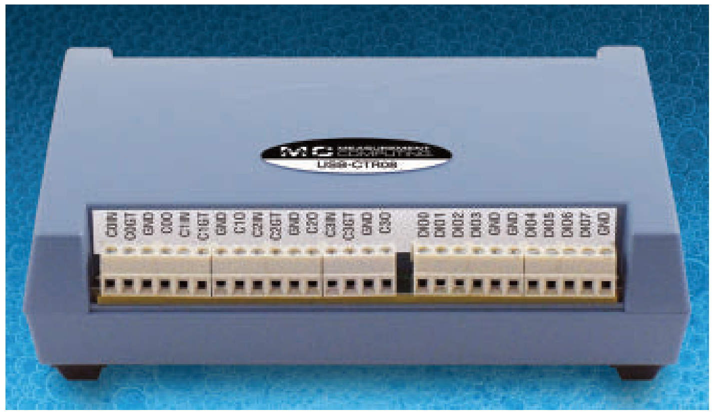
The brd-to-3D tool is a complement to CadSoft's EAGLE software that currently does not have a direct STEP export.

Customers can also obtain a free laser-sintered 3D model of the assembled printed circuit board with the purchase of a PCB-POOL prototype order.

For users not familiar with the 3D world, the PCB-POOL website provides video tutorials that explain handling and processing of 3D files to the photo-realistic renderings.

For more information, contact:
Beta LAYOUT, LTD.
Web: **www.pcb-pool.com**

USB COUNTER/TIMER DAQ DEVICES



Measurement Computing Corporation has announced the release of two high speed USB counter/timer DAQ devices available with four or eight counter I/O.

The USB-CTR Series features high speed pulse counting with a 48 MHz maximum input frequency. Resolution is programmable up to 64 bits, with an aggregate scan rate of 8 MB/sec.

These new devices offer some of the most advanced counter/timer functionality available in USB DAQ. Supported high level input modes include totalize, period measurement, pulse-width measurement, and timing measurement. Four independent PWM timer outputs and eight digital I/O are also provided. Timer output channels can operate continuously or for a specified pulse count, with duty cycle and period changeable on-the-fly. Counter and digital channels can also be scanned synchronously. Flexible edge, level, direction, and debounce settings allow the counters to better adapt to user signals.

USB-CTR Series features include:

- Four or eight counter I/O
- 48 MHz maximum input frequency
- Programmable resolution up to 64 bits per counter
- Aggregate scan rate of 8 MB/s
- Supports the following counter input modes:
  - Totalize
  - Period measurements
  - Pulse-width measurements
  - Timing measurements
- Debounce filter circuitry

- Four PWM timers
- Eight digital I/O
- Synchronous high speed reads of digital and counter inputs

Software options include comprehensive support for Visual Studio® and Visual Studio .NET, DASYLab®, and NI LabVIEW™.

The four-channel USB-CTR04 is priced at US$359, and the eight-channel USB-CTR08 is priced at US$429.

For more information, contact:
Measurement Computing Corp.
Web: **www.mccdaq.com**

RASPBERRY PI PROTOTYPING KITS

The surface-mount Raspberry Pi kits now available from Schmartboard each come with a Raspberry Pi through-hole add-on board and a choice of many Schmartboard SMT to DIP adapters which add the ability to use SC70, SOT, SOIC, QFN, QFN, and DFN components.

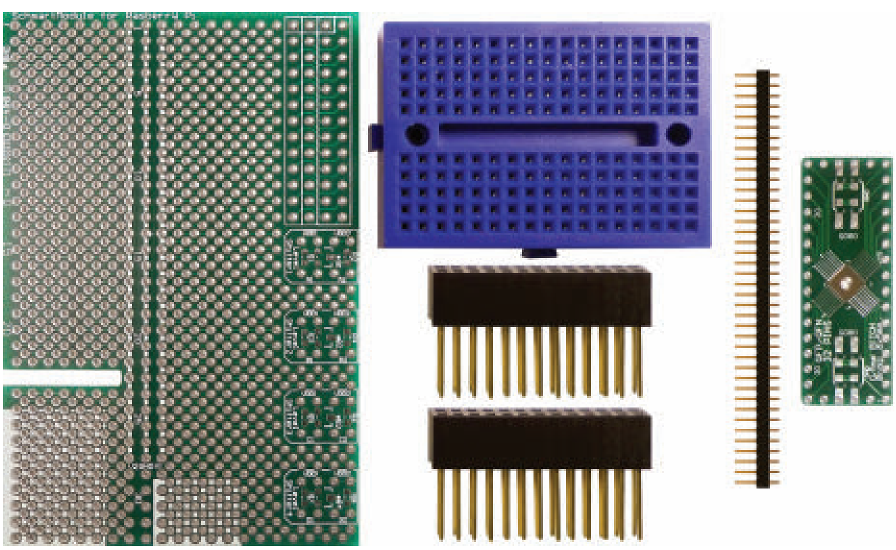
The Raspberry Pi base board features:

- An extra row of holes for easy access to the Raspberry Pi's general I/O signals.
- Rows of power and ground strips for easy power-up and flexibility.
- Pre-routed traces to minimize the use of wire jumpers.
- A slot for a video cable to keep the circuit clean and unencumbered.
- A marked area where the Raspberry Pi USB and Ethernet connectors are located to avoid conflicts.
- Headers with enough clearance to cleanly and safely stack on the board.
- Circuits for four level shifters.
- Schmartboard's signature offset through-hole grid which expands part placement options.

The kits are currently available in configurations to support SOT 23, SC 70, SOIC .5 mm, .635 mm, .65 mm, .8 mm, and 1.27 mm pitches, and many QFP, QFNs, and DFNs in both .5 mm and .65 mm pitches. More options will be added as which package types are most needed is determined.

The through-hole shields retail for US$13, bundled with the headers. The surface-mount kits retail for US$18, and additional SMT to DIP adapters retail for US$6.

For more information, contact:
SchmartBoard
Web: **www.schmartboard.com**



MCUs FEATURE ON-CHIP 12-BIT ADC, OP-AMPS, 16-BIT PWMs, and HIGH SPEED COMPARATORS

Microchip Technology, Inc., has announced an expansion of its eight-bit PIC16F178X enhanced mid-range core microcontroller (MCU) family with increased Flash memory densities, intelligent analog and digital peripherals such as on-chip 12-bit analog-to-digital converters (ADCs), 16-bit PWMs, eight-bit and five-bit digital-to-analog converters (DACs), operational amplifiers, and high speed comparators with 50 ns response time, along with EUSART (including LIN), I^2C and SPI interface peripherals.

The PIC16F178Xs are Microchip's first MCUs to implement the new programmable switch mode controller (PSMC) which is an advanced 16-bit pulse-width modulator (PWM) with 64 MHz operation and high performance capabilities. This combination of features enables higher efficiency and performance, along with cost and space reductions.

The new MCUs also feature eXtreme low power (XLP) technology for active and sleep currents of just 32 µA/MHz and 50 nA, respectively, helping to extend battery life and reduce standby current consumption.

Low power consumption in combination with advanced analog and digital integration make the PIC16F178Xs ideal for LED and other lighting applications, battery management, digital power supplies, motor control, and general-purpose applications.

Available in 28- and 40-pin packages, the MCU's intelligent analog integration paired with core independent peripherals — inclusive of the PSMC, DACs, op-amps, high speed comparators, and 12-bit ADC — enable self-sustaining smart control loops with minimal CPU intervention. This allows for optimal application control while freeing the CPU to provide incremental application value such as system health monitoring, communications, or human-interface control.

Initial evaluation and development of the PIC16F178X family can quickly begin with the F1 PSMC 28-pin evaluation board platform (US$19.99). Additionally, the PIC16F178X family is supported by Microchip's standard suite of development tools, including the PICkit™ 3 (US$44.95), MPLAB REAL ICE (US$499.98), and MPLAB ICD 3 (US$189.99).

For more information, contact:
Microchip
Web: **www.microchip.com**

1,200W MULTI-RANGE DC POWER SUPPLIES

B&K Precision has expanded its 9115 series with the addition of two new multi-range programmable DC power supplies: models 9115-AT and 9116. Like the 9115, the new models deliver full output power of 1,200W in any combination of voltage and current within the rated limits, and continue to offer the same features but with a few notable differences.

The 9115-AT provides unique built-in automotive test functions and the 9116 offers a higher voltage range up to 150V, making these new models suitable for automotive, as well as a variety of benchtop or ATE system applications.

The 9115-AT's special built-in automotive test functions can simulate common test conditions to ensure reliability of electrical and electronic devices installed in automobiles. Compliant to automotive standard's DIN 40839 and ISO 16750-2, these power test waveforms include: motor startup simulation, short voltage drop, reset behavior voltage drop, and engine start routine with cranking.

Front panel features from the 9115 carry over to the 9115-AT and 9116, which include a high resolution VFD display, independent voltage and current control knobs, cursors, and a numerical keypad for direct data entry. Both new models also provide internal memory storage to save and recall up to 100 different instrument settings, sequence (list mode) programming, and configurable overvoltage and overpower protection limits.

On the rear are standard USBTMC-compliant USB, RS-232, GPIB, and RS-485 interfaces supporting SCPI commands.

For ease of programming and remote control, the 9115 series now offers remote control software for front panel emulation, generation and execution of test sequences, and logging measurements via a PC.

The application software can also be integrated with NI's Data Dashboard for LabVIEW app (available for Android or iOS), allowing users to create custom

dashboards on smartphones and tablets for additional monitoring functions.

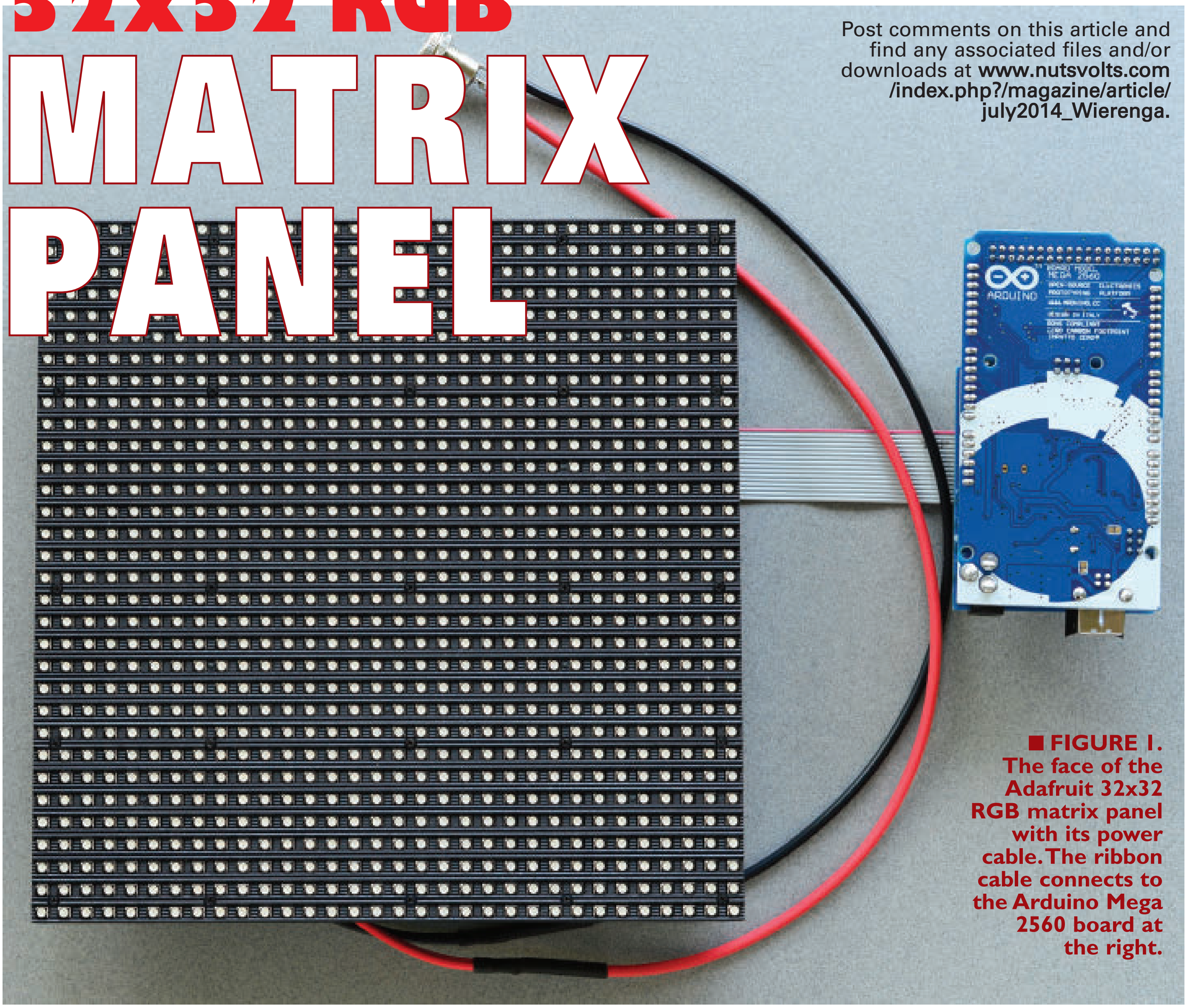
Available immediately, B&K Precision's 9115-AT and 9116 multi-range programmable DC power supplies are listed at a price of US$2,345 and US$1,995, respectively.

For more information, contact:
B&K Precision
Web: **www.bkprecision.com**

GIVING LIFE TO THE ADAFRUIT 32x32 RGB MATRIX PANEL

By Theron Wierenga

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com /index.php?/magazine/article/ july2014_Wierenga.**



■ FIGURE 1. The face of the Adafruit 32x32 RGB matrix panel with its power cable. The ribbon cable connects to the Arduino Mega 2560 board at the right.

Recently, I built a number of digital clocks in various configurations. Flashing LEDs have always interested me since I made my first digital clock many years ago with about 20 7400 series integrated circuits and red seven-segment outputs. My last digital clock displayed the time in different colored LEDs representing binary numbers for output. This got me thinking about an even more elaborate version — one with a matrix of 32x32 LEDs.

Scanning 1,024 LEDs is not a trivial matter — especially when you are going to use an Arduino Uno for all the computation. Output lines are restricted, and what one needs is some hardware to help with the scanning so the Arduino is free to do other things. While searching the Internet for ideas, I came across some red 8x8 LED matrix units that were mounted on a small board with a Maxim 7219 driver chip. These could be cascaded, and a decent software library for the Arduino was available to drive 16 of these units in a 32x32 array.

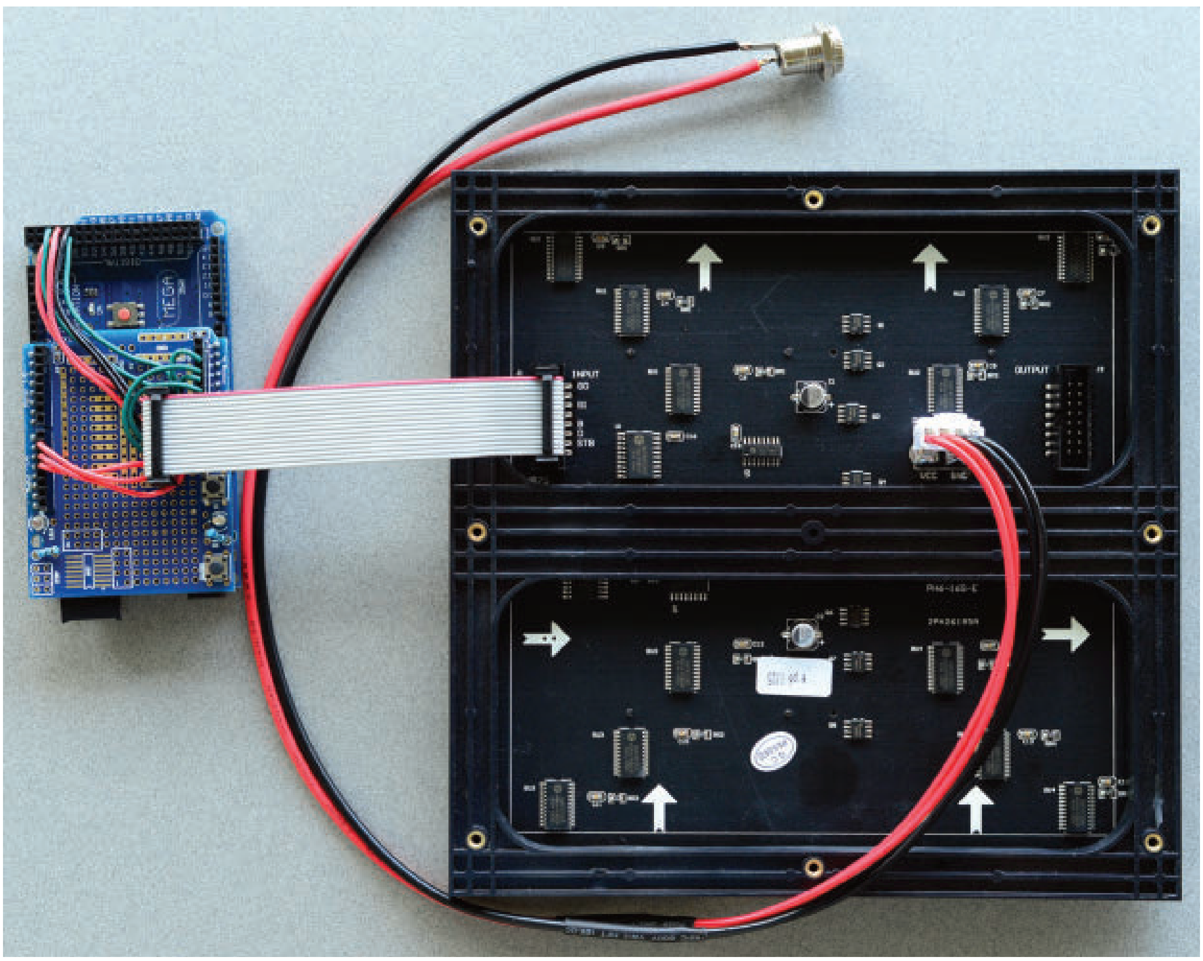
Just as this project was getting finished, I came across Adafruit's LED matrix panels. Their medium size 32x32 RGB panel looked really interesting. Their entire matrix was in one unit; it was RGB, so you have 3,072 LEDs to play with; they had supporting software, detailed instructions and examples on how to get the units wired and working; and the price was right.

The Game of Life

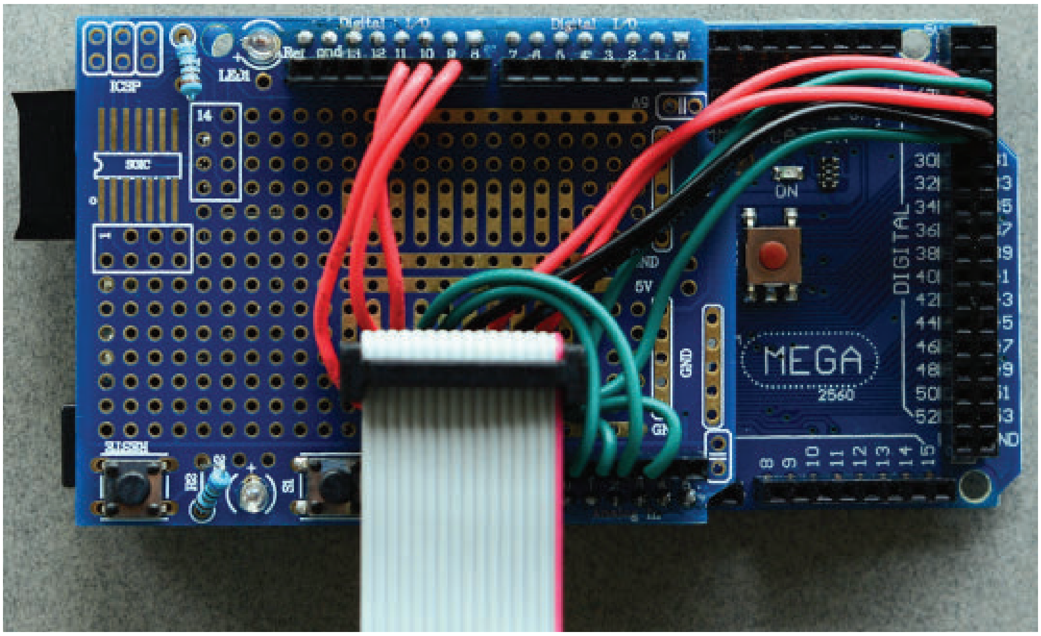
My imagination started to expand on the idea of putting a digital clock on one of these RGB panels. While thinking about how to add an interesting graphical display, the idea of adding Conway's Game of Life came to mind. If you are not familiar with the Game of Life, refer to the sidebar here or for more details, refer to the many web pages available on this subject.

Basically, the Game of Life is a variety of cellular automata that is generated using an array of cells. We will be using Adafruit's 32x32 panel of LEDs to represent these cells. You can begin by either illuminating a number of specific LED cells in a pattern, or just setting a random number of cells on. For the next generation, each cell and its eight neighbors are examined.

If a cell is on (LED illuminated), it will stay on if it has either two or three neighbors in the eight cells surrounding it. Otherwise, it will be off in the next generation. If a cell is off but has three neighbors that are on surrounding it, this cell will be on in the next generation. It is important to realize that one must create the new generation without any changes to the existing generation. Our Arduino sketch will need to maintain two 32x32 arrays to do this. What one sees as each generation is displayed is ever changing and flowing ornate patterns of cells that are simply created based on the previous generation.



■ **FIGURE 2.** The reverse side of the Adafruit 32x32 RGB matrix panel, showing its power cable plugged in and the ribbon cable connecting it to the Arduino Mega 2560 board using a blank project shield.



■ **FIGURE 3.** A close-up of the blank project shield used as a breakout board to connect the end of the ribbon cable to the Arduino Mega 2560 board.

The Panel

Let's take a look at the RGB panel from Adafruit. The panel comes in two different versions: one with two data connectors, and the other with just one. I received the version with one connector. The Adafruit website gives detailed instructions on how to connect the data plug to your Arduino, which can be either an Uno or a Mega. The directions are found at **http://learn.adafruit.com/32x16-32x32-rgb-led-matrix**.

The connector cable they supply is an IDC ribbon cable with connectors on each end configured as two rows of eight pins. I opted to connect my panel to an Arduino Mega 2560, not because the extra I/O pins are needed, but because it comes with 8K of SRAM instead of the Uno's 2K. Remember, there will be at least two 32x32 byte arrays of numbers representing the LED cells when computing a new generation in the Game of Life. More on this later. I used a prototype shield to make the data connections from the IDC ribbon connector to the Mega 2560. I installed a 2x8 header on the shield for the ribbon connector to plug into and then jumpered wires from this header to the various Mega pins. The Arduino pins used are different depending on whether you use an Uno or a Mega 2560, and whether the RGB panel has one or two data connectors. It is important to carefully follow Adafruit's online directions when making these connections.

Five volt power to the panel is provided by a separate connector. I purchased Adafruit's five volt/two amp power supply. It was a simple matter to cut off the spade connectors on the end of the panel's power cord and install a jack from RadioShack that mated with the 2.1 millimeter jack on the five volt power supply.

Test Time

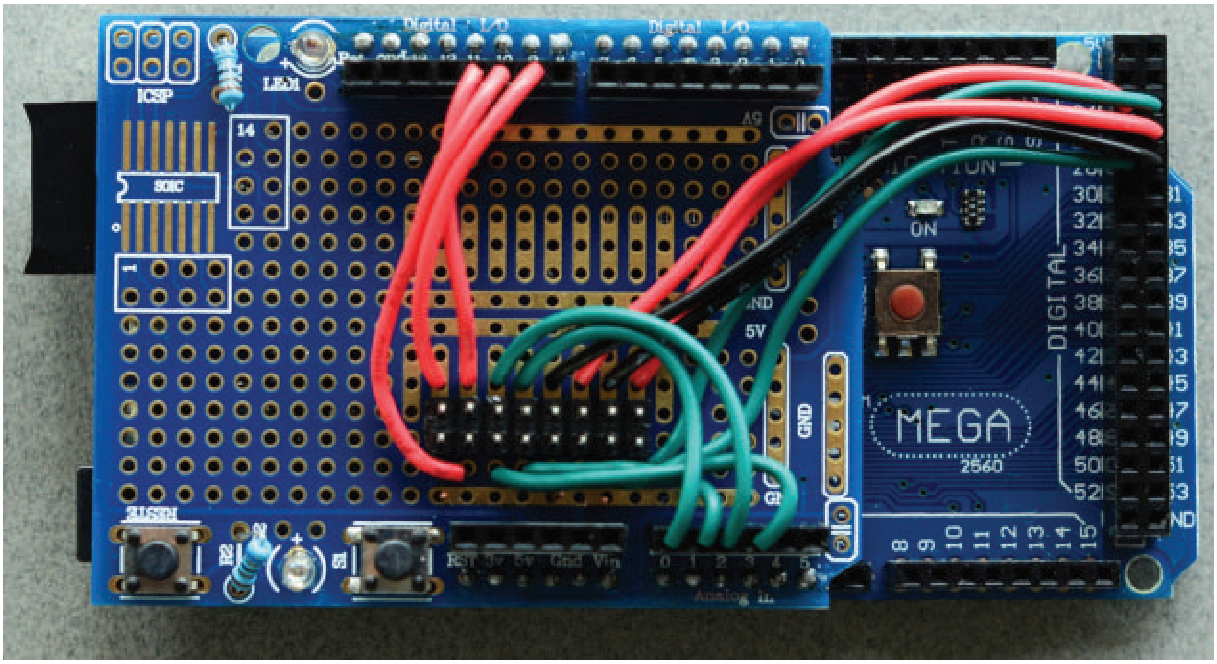
Once things are wired up, it is time to test the unit using one of Adafruit's example programs. There are two libraries you will need to install in the libraries folder of your Arduino software. These are *Adafruit_GFX*, the Adafruit core graphics library; and *RGBmatrixPanel*, with drivers specific to these RGB panels. The libraries are downloaded from the Adafruit site, and there are links to these libraries in the instructions for the panels.

I did find one glitch here. If you are running Arduino 1.0.5, you will get compile errors concerning the *Robot_Control* library. Adafruit advised me to either use Arduino 1.0.4 or temporarily remove the *Robot_Control* library from the libraries folder. I removed the *Robot_Control* library temporarily to the desktop and the example programs compiled and ran just fine. The example software in the *colorwheel_32x32* program contained all the information I needed to display my Game of Life program on the RGB panel.

In my program (available at the article link), I simply added the following lines — taken directly from their *colorwheel_32x32* example — to add the needed interface to the RGB panel:

```
#include <Adafruit_GFX.h>    // Core graphics
library
#include <RGBmatrixPanel.h> // Hardware-specific
library

// If your 32x32 matrix has the SINGLE HEADER
input,
// use this pinout:
#define CLK 11   // MUST be on PORTB! (Use pin 11
on Mega)
#define OE   9
#define LAT 10
#define A   A0
#define B   A1
#define C   A2
#define D   A3

RGBmatrixPanel matrix(A, B, C, D, CLK, LAT, OE,
false);

void Setup()
{
```

■ FIGURE 4. The breakout board with the ribbon cable removed. This shows the dual eight-pin header that is then connected to the appropriate pins on the Arduino Mega 2560 board.




```
matrix.begin();
}
```

Notice that I changed the clock pin (CLK) to pin 11 because I am using a Mega 2560. You must also choose the correct *#define* statements, depending on whether you have a single or double header for data input on the panel.

Once these statements are in place, you only need two lines of code in your program to turn on one of the LEDs in the panel. For example:

```
unsigned int c = matrix.Color888(0, 200, 0,
false);
matrix.drawPixel(x, y, c);
```

Color Methods

There are several Color methods provided with the matrix object: *Color333*, *Color444*, *Color888*, and *ColorHSV*. *Color333(7,7,7)* allows three-bit values for the red, green, and blue parameters; *Color444(15,15,15)* allows four-bit values, and *Color888(255,255,255*, false) uses eight-bit values, with an optional boolean parameter on the end to denote whether to use gamma correction on the color output. I simply set the gamma to false as I am only using a few colors.

All of these RGB color selector methods reduce the values to five-bit red, six-bit green, and five-bit blue to fit the panel's specifications. I found the *Color888* method a little buggy when values approach the 255 maximum; therefore, note my use of 200 in the previous example for a bright green color. The *ColorHSV* method uses the Hue/Saturation/Value color model, and takes one long variable for H, two byte values for S and V, and the boolean flag for gamma. H values should be in the range of 0-1535, and S and V 0-255.

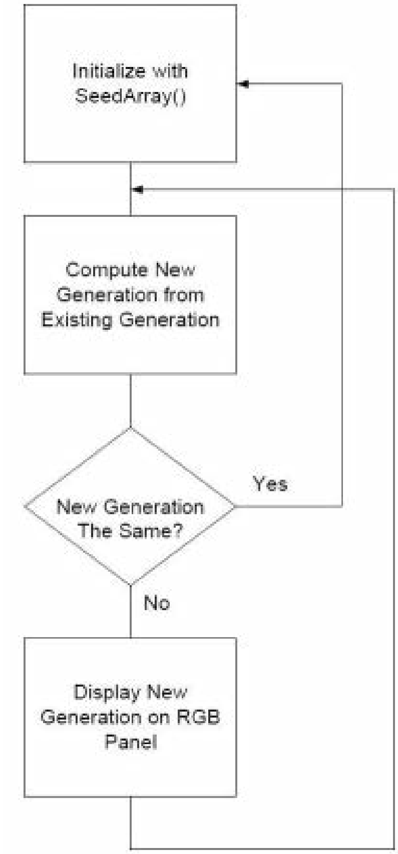
The *drawPixel* method is fairly self-explanatory. The LED at *x*, *y* (each value ranging from 0 to 31 on the 32x32 panel) is turned on when color *c* is created with one of the Color methods. Once an LED is turned on, it will remain on with the color chosen until changed in your program — as long as the panel is connected to the running Arduino. The *RGBMatrix* software takes care of scanning the panel in the background using the Arduino *interrupt*. This is a great feature as it greatly simplifies the code the user needs to write for operating the panel.

There are quite a number of other software methods demonstrated in the example software provided and interested readers can explore these for further information. There are methods to draw graphic lines, circles, rectangles, as well as text. With the *RGBMatrix* library, it is actually fairly easy to create complex output on the RGB panel. We have what we need to display the Game of Life by just picking a color and drawing a pixel (LED) with that color.

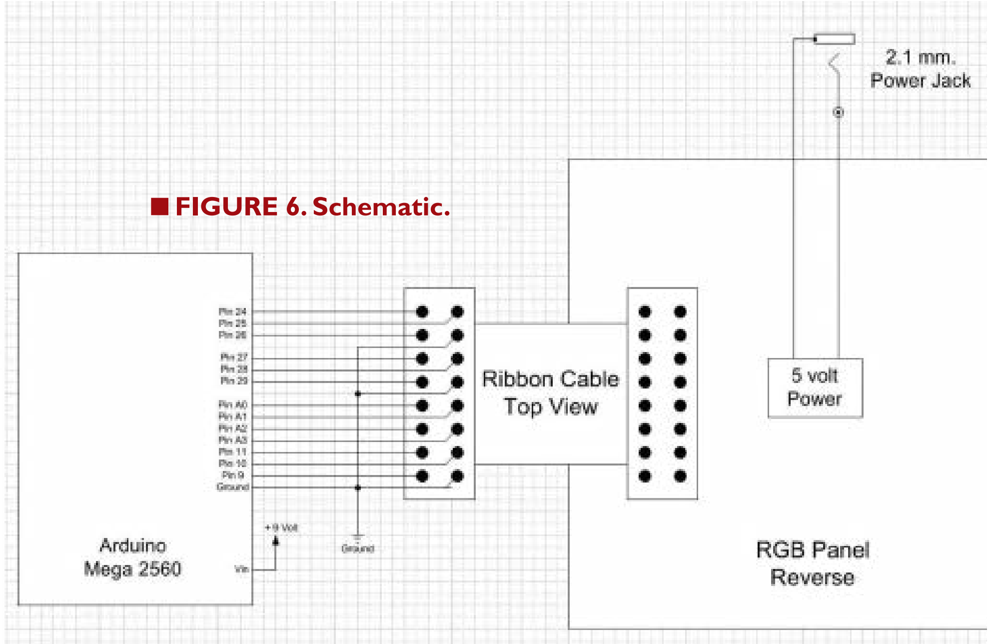
Previous Generation	New Generation	LED Color
0	0	LED off, an empty cell
0	1	Green LED, a new cell born
1	0	Blue LED, a cell dies
1	1	Red LED, cell continues to survive

Table 1.

■ FIGURE 5. Game of Life flowchart.



■ FIGURE 6. Schematic.



PARTS LIST

ITEM	SOURCE
RGB Matrix Panel, medium size	Adafruit #1484
Power Supply, five volt two amp	Adafruit #276
Power Jack, 2.1 mm	RadioShack 274-1563
Arduino Mega 2560	
Arduino Protoboard	
2x8 pin header or two 1x8 pin headers	
Hookup wire, solder	

Software

As we begin to think about software for the Game of Life, it becomes obvious that the model must be based on a 32x32 numeric array of numbers that represent the individual LEDs in the panel. However, because we must look at the eight surrounding numbers of each cell in the array, we run into a problem when trying to access cell members that surround the outside edges of the array.

Because of this, there are two approaches to solving this problem. One is to create a 34x34 array and just use the interior 32x32 set of cells, allowing the array to have a border of unused cells. These border cells are then set to be permanently off which has the effect of sometimes killing cells around the edges, but it does allow reading the surrounding neighbor cells without creating a negative array index. The second — and more involved approach — is to allow cell growth to wrap around the array. With this method, cells that move off the top of the array appear at the bottom, and the same occurs with cells along the sides. I chose the first method as it was easier.

The Game of Life program flowchart is shown in **Figure 5**. In my program, I simply created two arrays of 34x34 bytes using only the interior 32x32 array, set all the elements to zero, and randomly set 500 locations in the interior array to the value 1, with 0 representing an LED that is off, and 1 representing an LED that is on. I call this seeding the array.

Next, one does the computation for the next generation. One looks at the eight cells surrounding every cell in the 32x32 array, sums them up, and uses this sum to determine if the cell in the new array of 32x32 is set to a 0 or 1 by the Game of Life rules outlined previously. After the computations for a new generation are finished, the new generation is written to the RGB panel. Because I wanted to utilize the additional colors available in an RGB display, I decided to make the display of each generation a little more complicated. Here is what I did.

This adds a lot more color to the display. Because of the speed of the Arduino and the speed of the display, one really needs to put a delay in after the new generation is displayed. If this is not done, the display moves so fast the interactions are impossible to see. Timing things, I found that a new generation is displayed about every 50 milliseconds with no delay.

Next in the flowchart, we have to discover if there have been any changes from one generation to the next. If the generations are the same, the cells in the Game of Life have become stable and the display will not change. This is pretty boring, so we simply restart the process with a new random seed to the array. If there have been changes from

There is a second program available at the article link that packs the Game of Life values into single bits. This allows the arrays to be much smaller, and it can check back eight generations for oscillators. This does add a lot of overhead, however. A generation now takes about 1.4 seconds to update, but it is still fast enough to make the display interesting.

one generation to the next, we copy the new generation array to the current generation array and then begin the computation for a new generation.

Keeping It Fresh

Actually, things get a little more complicated here. If you do a little reading about the Game of Life, you soon begin to realize that there are some simple repeating patterns that can occur every 2, 3, 4, 5 ... generations that are called oscillators. Two generation oscillators are quite common, and perhaps the most common is a pattern called traffic lights. This is a pattern of three living cells in a horizontal row surrounded by empty cells. In the next generation, this becomes a vertical row of living cells; the two end cells die because they only have one neighbor; and two cells are created above and below the center cell because it has three neighbors.

In the next generation, it returns to a horizontal row of three living cells and then flips back and forth with each new generation. If the display has only stable cells in patterns that do not change — like a square block of four, and some traffic lights — then the display will just switch back and forth between the two. Again, very boring, so we need to detect this and restart the process.

In order to do this, we must remember back an additional generation, i.e., we will need to store three generations in three 32x32 arrays. Then, we not only need to compare the new generation to the current one, but also go back and compare it to the previous one. If the new generation is the same as either one of the last two, we restart the process. If we want to check for three generation oscillators, we would need four arrays; four generation oscillators, five arrays, etc.

Unfortunately, the Mega 2560 has only enough memory for three generation oscillators, but not four. The problem here is that we are being wasteful and storing our two-bit colors representing off, red, green, and blue in an eight-bit byte. This simplifies our programming task, but restricts the number of generations back we can check. It is possible to store four two-bit cells in a single byte, but I will leave that program model to those readers who are interested enough to write the necessary code.

Wrap-Up

As mentioned, the complete Arduino sketch is included in the download material and contains lots of comments. This project began with thinking about a new display for a digital clock and morphed into a Game of Life display. For the future, there is still a plan to add a real time clock circuit to the Arduino shield and alternately display the time and date interspersed with a display of the Game of Life to keep things lively. **NV**

The Game of Life

The Game of Life — also known simply as Life — is a cellular automaton devised by the British mathematician, John Horton Conway in 1970.

"Life" is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves.

The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states: alive or dead. Every cell interacts with its eight neighbors, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

- Any live cell with fewer than two live neighbors dies, as if caused by under-population.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if by overcrowding.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed — births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick (in other words, each generation is a pure function of the preceding one). The rules continue to be applied repeatedly to create further generations.

Conway was interested in a problem presented in the 1940s by mathematician John von Neumann, who attempted to find a hypothetical machine that could build copies of itself. He succeeded when he found a mathematical model for such a machine with very complicated rules on a rectangular grid. The Game of Life emerged as Conway's successful attempt to drastically simplify von Neumann's ideas.

The game made its first public appearance in the October 1970 issue of *Scientific American*, in Martin Gardner's "Mathematical Games" column. From a theoretical point of view, it is interesting because it has the power of a universal Turing machine; that is, anything that can be computed algorithmically can be computed within Conway's Game of Life.

Gardner wrote:

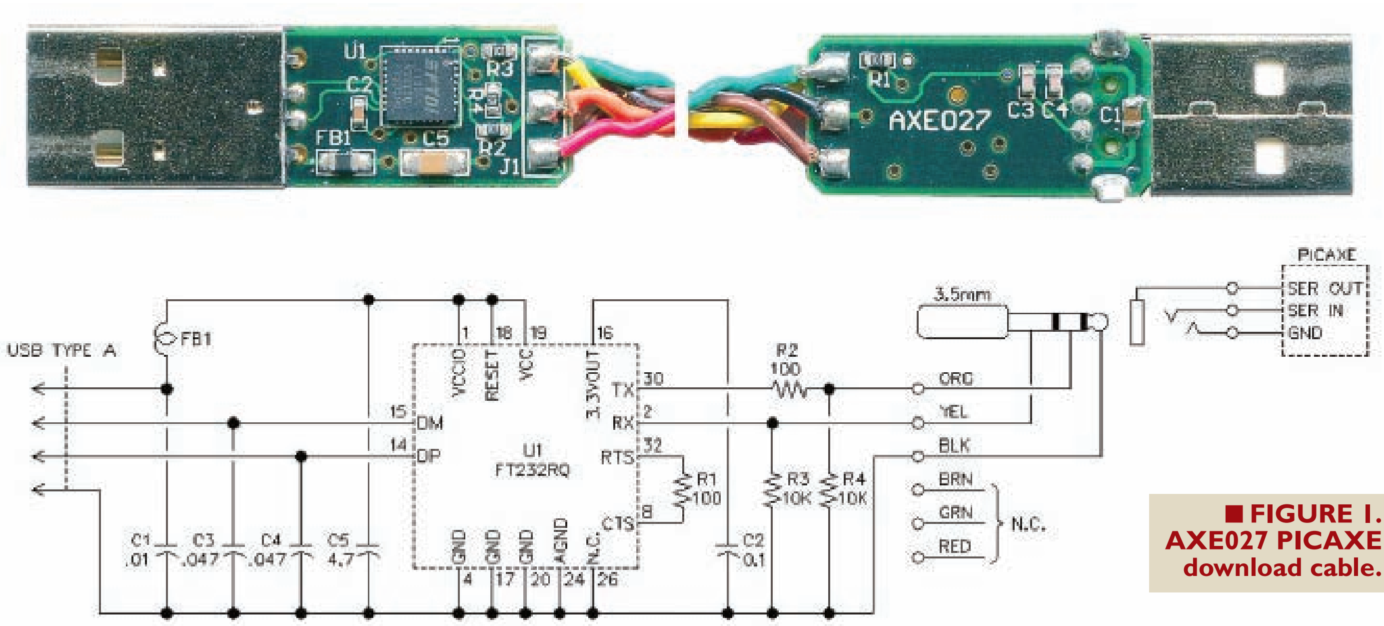
The game made Conway instantly famous, but it also opened up a whole new field of mathematical research — the field of cellular automata. Because of Life's analogies with the rise, fall, and alterations of a society of living organisms, it belongs to a growing class of what are called "simulation games" (games that resemble real life processes).

Courtesy of ***en.wikipedia.org****.*

BUILDING YOUR OWN PICAXE DOWNLOAD CABLE

By Tommy N. Tyler

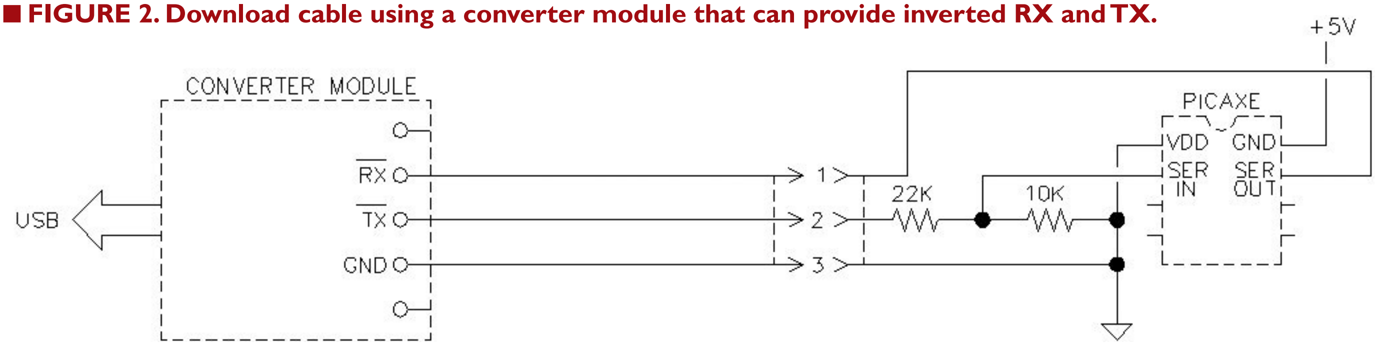
Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/index.php?/magazine/article/july2014_Tyler.**



■ **FIGURE 1. AXE027 PICAXE download cable.**

PICAXE products offer an attractive alternative to novices who lack experience using microprocessors and would like to learn more about them without a major investment in development tools and software training. PICAXE chips are Flash-programmed PIC processors containing a proprietary interpreter that offers powerful, easy-to-use, BASIC-like instructions. The chips are inexpensive, and the editing software for writing and testing the program is a free download. Ample info and tips are available in forums and published manuals. (And don't forget about *Nuts & Volts* bi-monthly column, the PICAXE Primer!) What else could you want?

■ **FIGURE 2. Download cable using a converter module that can provide inverted RX and TX.**



Many hobbyists are reluctant to venture into new technology because of startup costs. A PICAXE provides a low cost way to get started with microprocessors. There is one catch, though. You may have assembled your first circuit on a solderless breadboard using only a few dollars in parts and written and debugged a program with the fun and exciting simulator built in to the free programming editor, only to discover that transferring the program from your PC into the PICAXE chip required a special USB download cable that costs $20 to $30. This article addresses that issue.

The PICAXE system uses a serial comm cable to send a program from the program editor in a PC directly into the chip. No other programming equipment is required. PICAXE's AXE027 download cable has a USB-to-TTL RS-232 converter circuit on a small printed circuit board (PCB), which is embedded within a molded USB Type A connector at one end and a 3.5 mm miniature stereo plug at the other end. The PCB is a bit of an overkill, with top, bottom, and intermediate ground plane layers. **Figure 1** shows details of the PCB assembly removed from a recently acquired AXE027 cable and its schematic. This assembly is similar to an older design shown in PICAXE documents, but it has a few minor differences.

The cable uses an FT232Rx series converter — a chip found in many USB-to-serial converters because of its versatility, as well as the availability of drivers for a wide variety of operating systems. One reason PICAXE chose this chip was because of its compatibility with the company's original serial comm system. FTDI's converter chips allow you to invert the TX and RX signals so they are active low, just like standard RS-232 signals.

When you look at the chip's configuration EEPROM with FTDI's utility program (*FTProg 2.6.8*), the descriptive data for the AXE027 is:

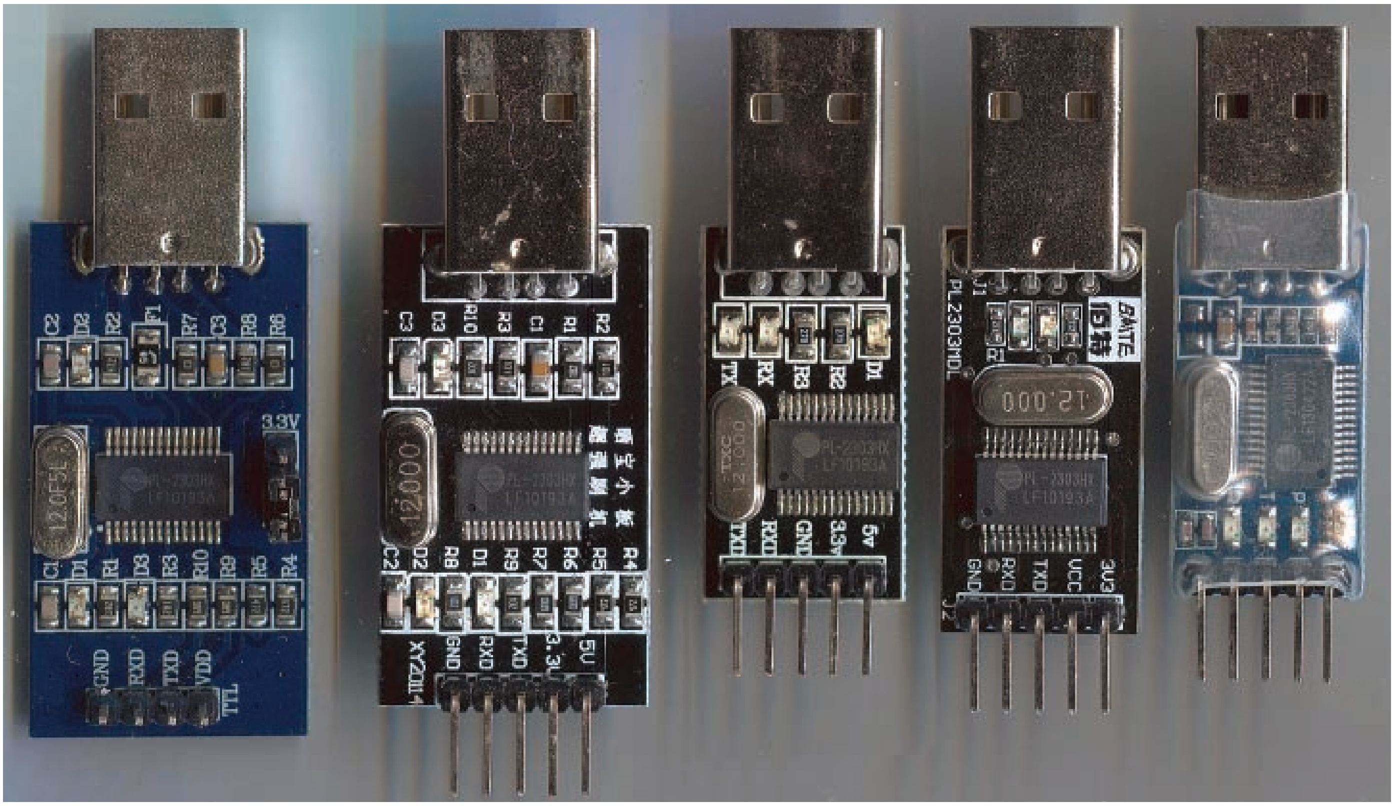
Chip Type: 'FT232R'
Vendor ID: 0403
Product ID: BD90
Product Description: 'AXE027 PICAXE USB'

Under hardware-specific invert RS-232 signals, *FTProg* shows that TXD, RXD, RTS#, and CTS# were all inverted. (Output designations are followed by a # symbol if they are active low by default.) Note that RTS and CTS are jumpered together through R1 on the board. Inverting TXD and RXD is absolutely necessary, but other cables I have tested seem to work just fine with or without inversion of RTS# and CTS# — regardless of whether they are connected. Also, some PICAXE documentation shows a 100 ohm resistor in series with RX — like R2 in series with TX — but my purchased sample has no such resistor nor any place for it.

FTProg also shows that all five CBUS outputs of the AXE027 cable were reconfigured from their FTDI factory default configurations to TXDEN. I don't know why that was done, but it has nothing to do with using the cable to download PICAXE programs. In summary, there are ways that PICAXE could have made it difficult to substitute third-party download cables, but to their credit there is no evidence they have attempted to do that.

The requirements for a USB download cable for PICAXE are few and simple: just three wires — serial in, serial out, and ground. The key component is a USB-to-RS-232 converter chip that can operate at 5 VDC and 4800 baud (the PICAXE comm frequency). There are at least four major suppliers of these chips worldwide: FTDI, Prolific, SI Labs, and Microchip; some can provide the inverted RX/TX configuration. For the others, you can add a couple of simple inverters. Purchasing a ready-made converter module or breakout board can save you the tedious job of soldering the small closely-spaced pins of a converter chip.

Figure 2 shows a basic download cable circuit using a module with an FTDI 232 series or Microchip MCP2200 chip, either of which allows you to invert the RX/TX signals by modifying the internal configuration of the EEPROM. I've shown a generic three-pin connector rather than the stereo connector used by PICAXE. The 22K and 10K resistors cannot be part of the cable. They keep the PICAXE serial input from floating around like an antenna

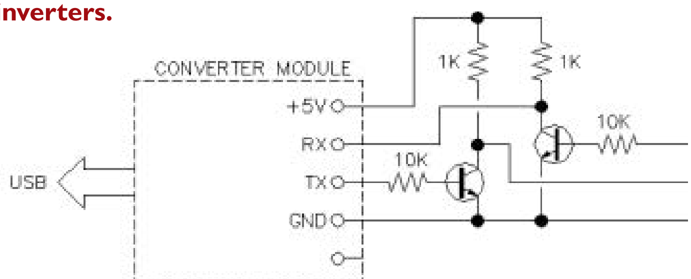


■ **FIGURE 3. Examples of Prolific and SI Labs USB modules.**

when the download cable is disconnected, which could result in noise pickup that might interfere with program execution. Their importance is clearly explained in PICAXE documentation.

If you want your download cable to be really small, take a look at Jim Paris' FTXUSB serial breakout board (**www.tindie.com/products/jimparis/microftx-usb-serial-breakout-1/?pt=directsearch**). At only 0.4" x 0.64" — including a micro USB B connector — it's the smallest cable currently being made. It uses a tiny FT230XQ chip with RX and TX signals that you can invert with *FTProg*. The cost is a little over $12 shipped, to which you must add a USB micro cable and an output cable.

■ **FIGURE 4. Download cable with transistor inverters.**



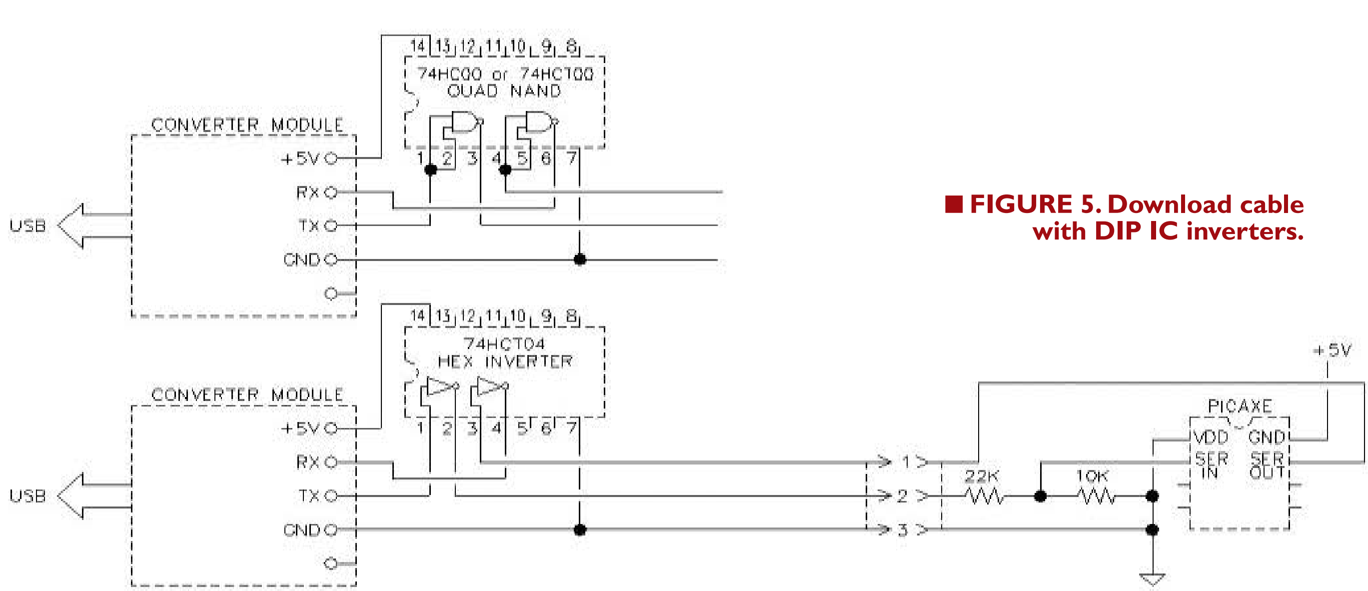
The next largest category is the CRIUS FTDI basic breakout board from DealExtreme (**www.deal extreme.com**). At only $6.59 (including shipping), this high quality FTDI board also doesn't need extra inverters. All you would need to do would be to install input and output cables and use *FTProg* to invert RX and TX.

Omega MCU Systems sells a board with an SI Labs CP2103 chip and a couple of transistor inverters added (**www.onebytecpu.com/usermanuals/u2p_datasheet.pdf**), but it costs about $15 shipped, and you would still need cables. It's also a little clunky.

You can buy a postage stamp-sized MCP2200 breakout board direct from the manufacturer (Microchip) or from major electronics suppliers (Digi-Key, Mouser, Newark, Farnell, etc.) for $15 plus shipping. Microchip provides an EEPROM configuration utility that is similar to *FTProg* that allows you to invert the RX and TX signals. All you would need to add would be cables.

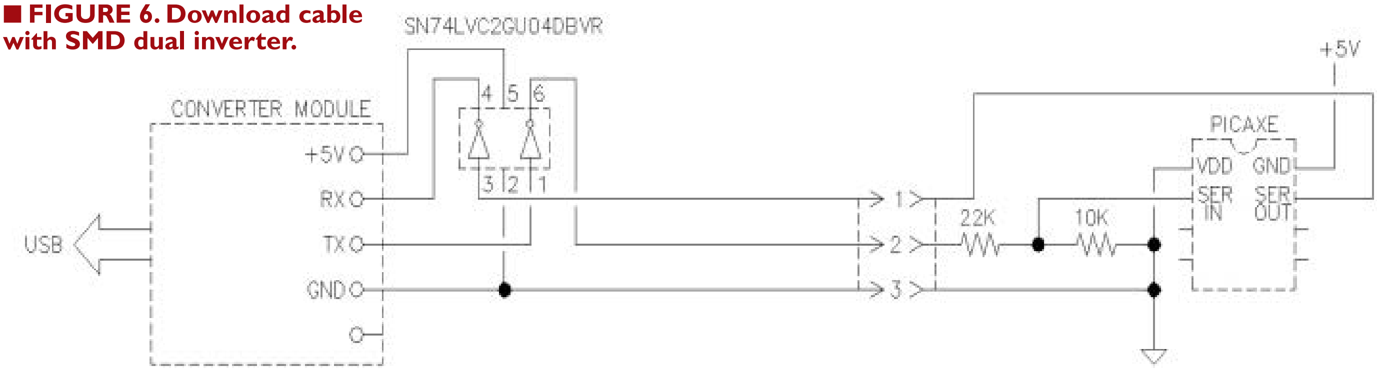
For the lowest cost download cable, you can use a USB-to-TTL converter module based on a Prolific PL-2303HX chip or an SI Labs CP-2102 or 2103 chip. These are available from many sources in China, Taiwan, and Hong Kong. **Figure 3** shows some examples.

An easy way to recognize a board using a Prolific or SI Labs chip is the 12 MHz crystal. (FTDI-based boards



■ **FIGURE 5. Download cable with DIP IC inverters.**

■ **FIGURE 6. Download cable with SMD dual inverter.**



use a built-in oscillator, so no external crystal is needed. The MCP2200 board uses a tiny 12 MHz ceramic resonator.) Most modules have five output pins (RX, TX, GND, +3.3V, and +5V), giving you a choice for powering other circuits. Some boards (like the one on the left) only have four pins, with a jumper block for selecting Vdd (+3.3V or +5V).

There is no standard pinout sequence. You can temporarily power your PICAXE circuit from the module, but for a final design use something that works when the download cable is unplugged.

Nearly all modules have one or more LEDs to indicate power, transmit, receive, etc. All of them use a USB Type A connector which eliminates the need for a separate USB cable. My favorite is the board at the far right of **Figure 3**, which is supplied with a protective outer wrapper of clear heat shrink tubing. It uses a Prolific PL-2303HX chip, is well built, reliable, and has good stable drivers.

If you search Google for CEG004200 or search eBay for PL-2303HX, you'll find it widely available for as little as $1 or $2 including shipping. None of these boards have the capability of inverting RX and TX. You must add a couple of inverters powered by the module's +5V output. That's easily done in several ways.

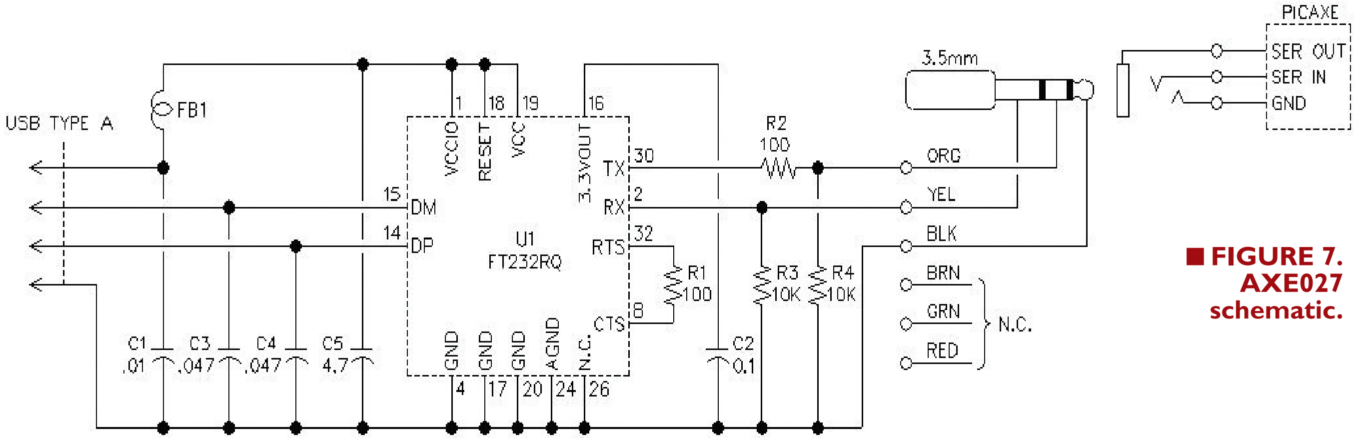
Figure 4 shows how you can use discrete transistors and resistors for the inverters. Any type of NPN transistors can be used, and the resistor values are not critical. Note that the 22K and 10K resistors from **Figure 2** are still necessary to prevent noise on the serial input when the cable is disconnected.

Transistor inverters are inexpensive and parts are readily available. However, if you want to simplify your cable with just one component instead of six, you can use almost any digital IC with two or more inverters, such as the 74HCT04 hex inverter or the 74HCT00 quad NAND shown in **Figure 5**.

There are many choices of similar ICs in a 14-pin DIP package with through-hole leads, and many hobbyists prefer this type of package because it is easy to work with when using perfboard or a solderless breadboard.

If you can deal with surface-mount components and want something elegant, a good choice is the SN74LVC2GU04DBVR in a tiny SOT-23 package with just six pins as shown in **Figure 6**.

To package the additional inverter circuitry, attach a small perfboard extension to the end of the module board



■ **FIGURE 7. AXE027 schematic.**

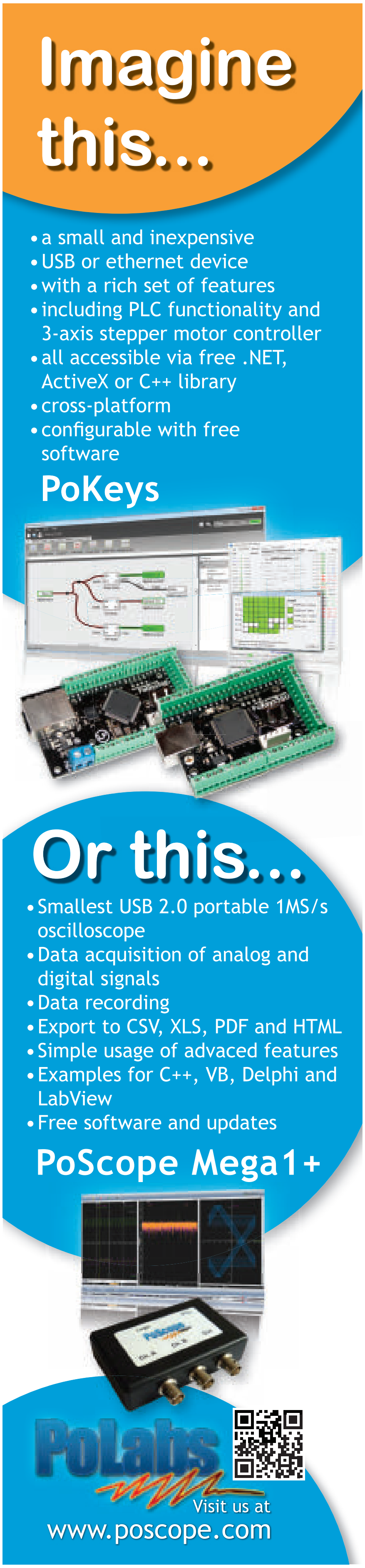
with epoxy. For connections to the inverters, use tiny #30 gauge wire-wrap wire or strands from stranded hookup wire. After anchoring a three-conductor cable and testing the unit, cover the add-on board assembly with epoxy or heat shrink. To test your completed cable — including the inverters — jumper pin 1 to pin 2, and then use Windows HyperTerminal (or any other serial comm utility software) to perform a loopback test.

If you are using a PICAXE development board, you can terminate the cable with a standard 3.5 mm stereo plug or any suitable three-conductor connector. Since most of the converter modules will operate up to 115 Kb, you can use your cable for many computer interface projects other than PICAXE downloading. Most other applications for a serial comm cable use standard TTL polarity signals, so you won't need the inverters. **NV**

PARTS LIST

Here's a summary parts list that applies to the simplest versions of the cable. Select the miscellaneous parts per the version you are building.

Prolific PL-2303-HX Converter Module	
2N3904 NPN Transistor	Digi-Key Part Number 2N3904FS-ND
1/8W Resistor 1K	Digi-Key Part Number CF18JT1K00TR-ND
1/8W Resistor 10K	Digi-Key Part Number CF18JT10K0CT-ND
74HCT04 Hex Inverter	Digi-Key Part Number 296-1605-5-ND
74HCT00 Quad NAND	Digi-Key Part Number 296-1603-5-ND
SN74LVC2GU04DBVR Dual Inverter	Digi-Key Part Number 296-13276-1-N


Imagine this...
• a small and inexpensive
• USB or ethernet device
• with a rich set of features
• including PLC functionality and 3-axis stepper motor controller
• all accessible via free .NET, ActiveX or C++ library
• cross-platform
• configurable with free software
PoKeys
Or this...
• Smallest USB 2.0 portable 1MS/s oscilloscope
• Data acquisition of analog and digital signals
• Data recording
• Export to CSV, XLS, PDF and HTML
• Simple usage of advaced features
• Examples for C++, VB, Delphi and LabView
• Free software and updates
PoScope Mega1+
PoLabs
Visit us at
www.poscope.com


HEXBRIGHT
The
Brightest
Most Durable
Arduino
Flashlight.
Period.
hexbright.com


EARN MORE MONEY
Get your dream job!
Be an FCC Licensed Wireless Technician!
Make up to $100,000 a year and more with NO college degree
Get your "FCC Commercial License" with our proven Home-Study Course!
• No costly school. No classes to attend.
• Learn at home. Low cost!
• No experience needed!
• MONEY BACK GUARANTEE: You get your FCC License or money refunded!
The United States of America
Move to the front of the employment line in Radio-TV, Communications, Avionics, Radar, Maritime and more... even start your own business!
Call now for FREE info: 800-877-8433 ext. 109
www.LicenseTraining.com
COMMAND PRODUCTIONS FCC License Training
Industrial Center, 480 Gate Five Rd., PO Box 3000, Sausalito, CA 94966-3000

BUILD THE CAT AWAY!

By Alan Parekh

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/index.php?/magazine/article/july2014_Parekh.**

We recently added a new kitten to our household. Little Chubby is adorable, but he is still learning what is off limits. We don't allow our cats (we now have two) on the tables, and this random rule seems to be hard for cats to learn since they love to jump up on anything that isn't their cat house. Our method of behavior reinforcement is shooing them off whenever we catch one sitting on a table. Of course, this method takes a long time since they might have been sitting there for hours before you catch them. They are also quite smart, often they hear you coming and jump off before you catch them in the act.

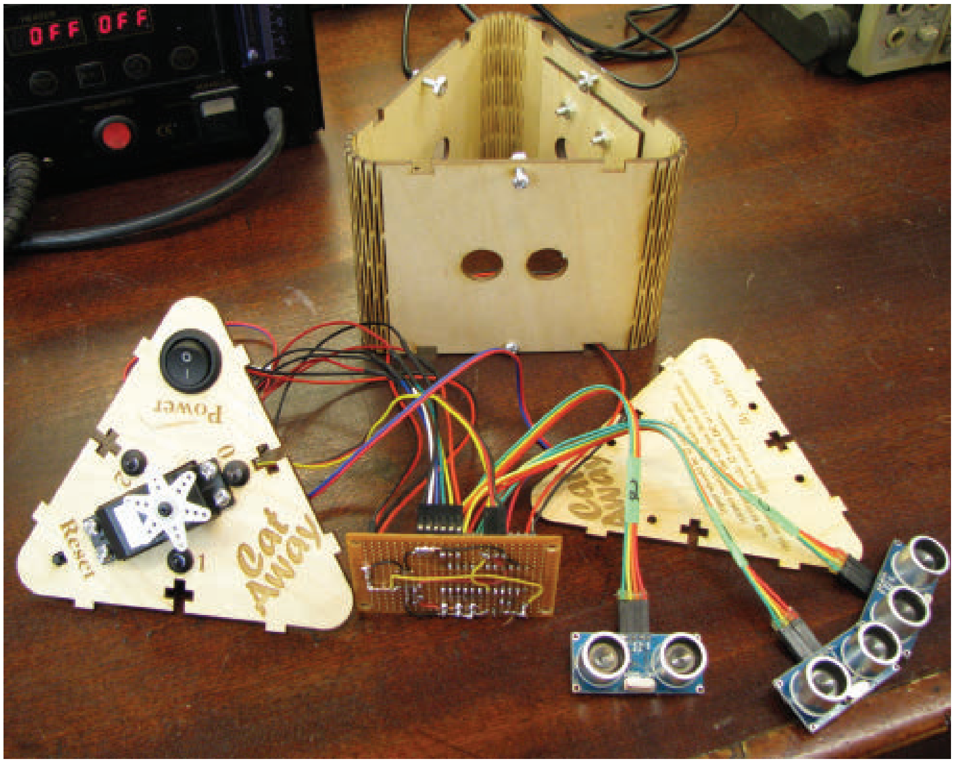
Solution

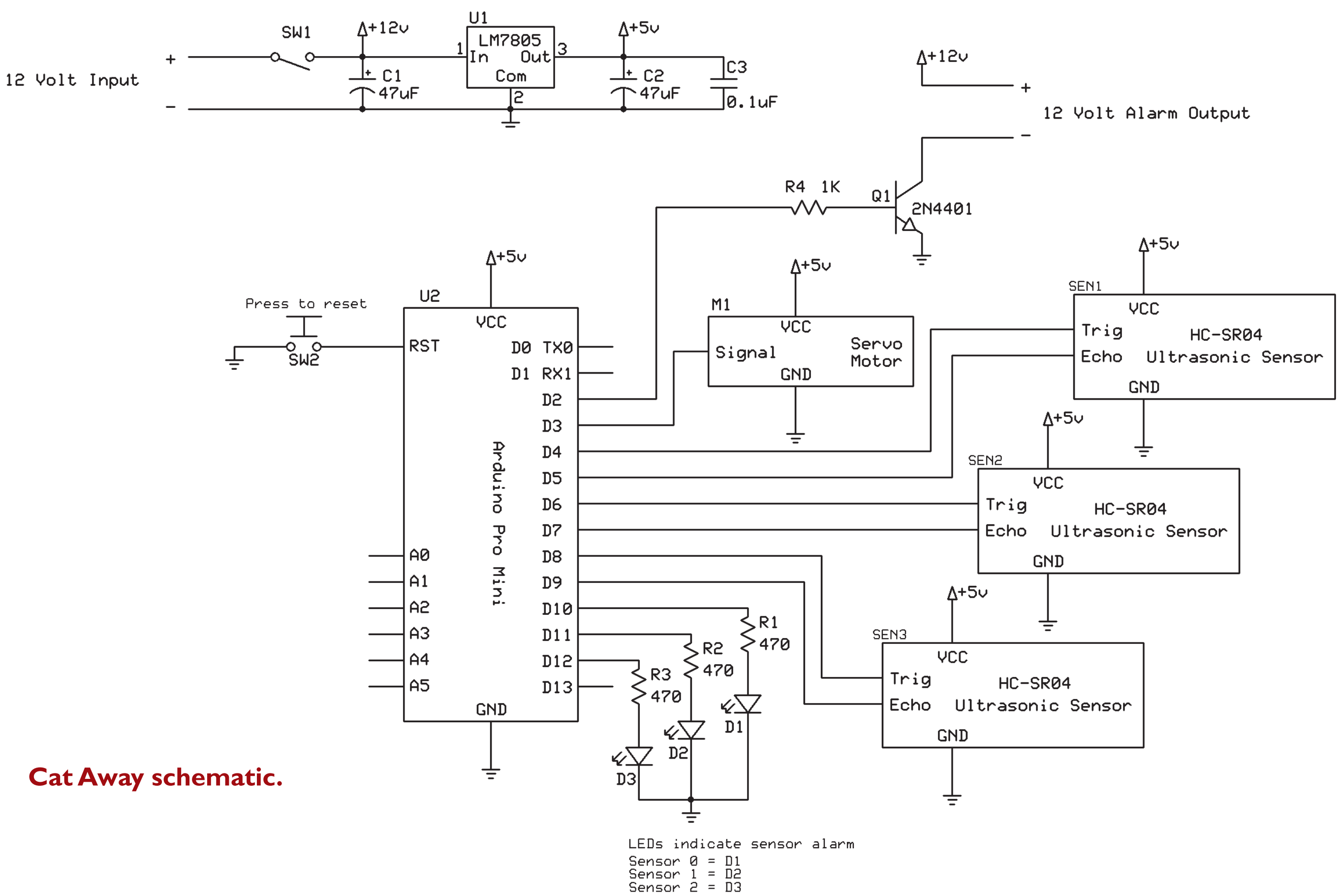
I thought this would be a great problem to solve with a new electronic project. I envisioned a small box that could be moved from table to table and would automatically enforce the rules.

The build constraints would be to construct it inexpensively and in a timely manner. This basically meant that it would need to be made from parts that were on hand since shipping things in from most places takes a few weeks.

Detection

My first thought was to use a motion sensor, but it would need to be placed in such a way that it wouldn't send false alarms whenever we were walking around the table. Infrared beams were another idea, but it would require IR LEDs and IR





Cat Away schematic.

Power button.

Reset button.

Screw nut.

"Twirling thing."

transistors to be mounted in inconvenient places on the table. I guess if there was no time or real budget, a cool over-the-top method would to use a ceiling mounted Microsoft Kinect sensor to generate a 3D representation of the table and use software to look for a cat. The optimal solution within budget was an ultrasonic sensor which could be set up so it is looking out across the table. Then, it could measure the distance between it and objects in front of it. This way, it can determine if the object is close enough to be concerned with or if it should be ignored.

Alarm Method

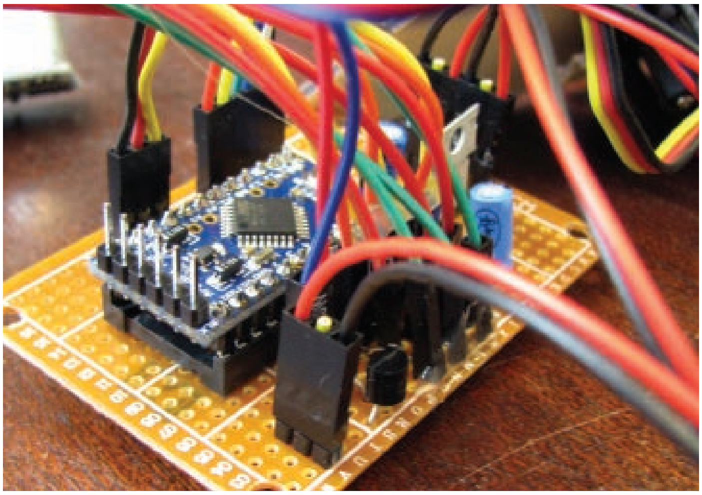
Once you have detected that the cat is on the table, you need to scare it off. There were three motor types that were considered: a gear motor, a servo motor, and a stepper motor. All would have done the job, but some would have taken more work. The gear motor would turn slow enough and have enough torque to turn something like a small doll (my cat version of a scare crow); the issue is that the gear motor can stop at any location and would need some type of index sensor to allow the actual position of the doll to be known. This is required since the arms of the doll could possibly cover a sensor if it came to rest in the wrong rotational position.

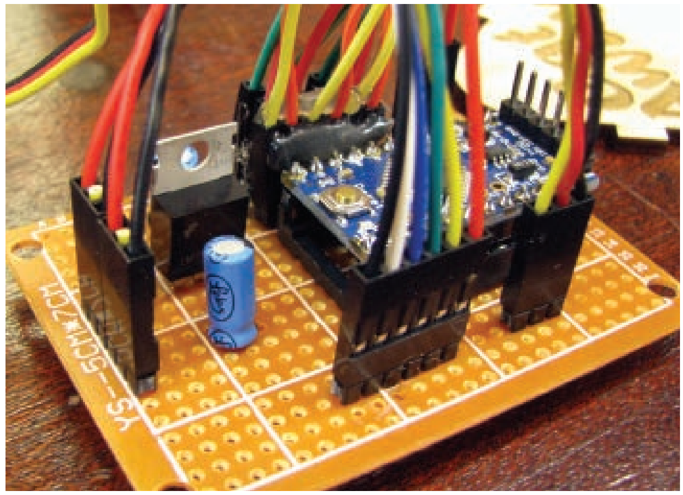
The stepper motor was ruled out since when powered down, it could be turned out of position. So, even if it didn't lose any steps during operation, a true starting position would still be needed. The servo was selected because of its simple wiring requirements and that no external position sensor was required. The servo can easily be commanded to move to a certain position, and it works quickly and reliably. Best of all, only three wires are needed: 5V, ground, and a signal from the microcontroller.

A second alarm method for easily adding different scaring abilities is a 12 volt alarm output. With this output, you are no longer limited to the movement (and sound) of the servo. A loud 12 volt buzzer or vibration motor would be a simple addition to the system to guarantee results.

Project Box Selection

The design was going to be stuffed in an off-the-shelf project box, which meant it would have four sides. So, it would need four ultrasonic sensors. I thought this would be a bit of overkill since cats usually snoop around for a bit before they settle down for a rest on the table. In my mind, I was thinking three areas of detection from a box mounted





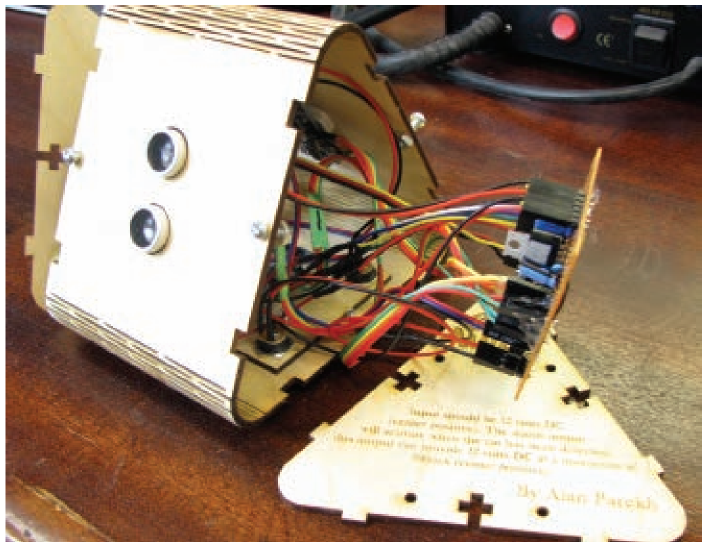
in the center of the table would be ideal. The only issue with that is I have never seen a triangular project box and I certainly didn't have one on hand. The plan to expand the build to a custom enclosure was made so that three sensors could be used.

Circuit

The heart of this project will be the microcontroller. Since the task isn't complex, almost any microcontroller would do. The main thing I was concerned about was that I had enough I/O pins to handle everything. I chose the Arduino Pro Mini since I had some on hand, I love the small size, they have a ton of I/O pins, and the price is such that you would simply use a new one for the next project rather than scavenge one from an old one. As a bonus, this microcontroller is programmed using an FTDI cable which makes it very easy to do serial port debugging. The HC-SR04 ultrasonic sensors have four connections: two for power; a trigger; and an echo pin. The trigger and echo pins are connected to the microcontroller to be able to initiate a measurement and get the results. There are some ultrasonic sensors such as the Parallax Ping))) that require only one I/O pin. So, if I was I/O constrained, the Parallax version could be an easy way to lighten my I/O needs.

Next, we have three LEDs that will be used to indicate which sensor was triggered. These three blue LEDs will be driven directly from an I/O pin since they will only draw around 4 mA with the 470 ohm current-limiting resistor. The LEDs and resistors are shown as individual components in the **schematic**, but I ended up using some that were pre-built with the resistors soldered in place and leads attached.

The servo will use just one I/O pin to receive the position information it needs. The servo was selected such that it would have enough torque to spin something fairly heavy. The Dynam B2232 that was used has a torque of 44.4 oz-in at 4.8 volts. The use of a servo motor meant that using the voltage regulator that is built into the Arduino Pro Mini would be out of the question, and that a stand-alone

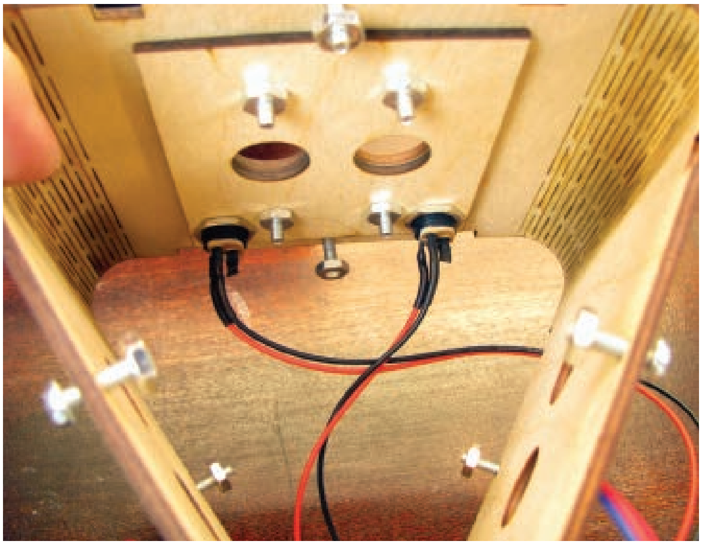


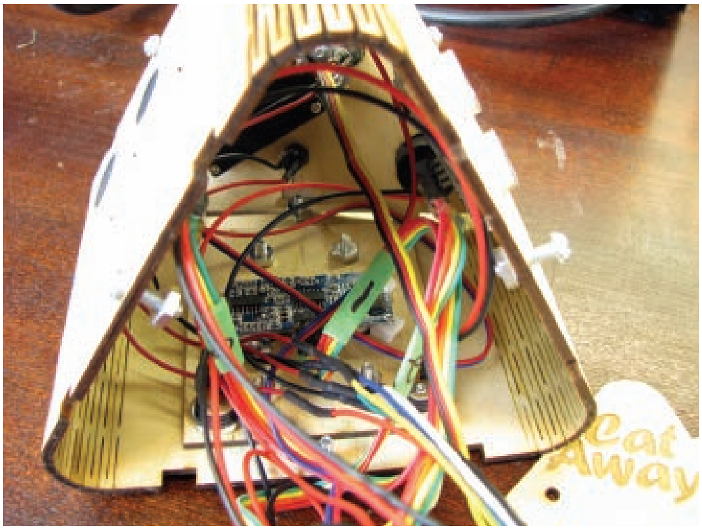
voltage regulator would be needed.

The power supply section is very simple. A 12 volt DC plug-in power supply will give power to the box using a barrel connection; a LM7805 in a TO-220 package along with three filter capacitors will be used to generate the five volts for the electronics. A power-on switch will be wired in series with the power-in and the feed to the circuit so that the circuit can be powered down completely when needed.

The 12 volt alarm output is buffered using a small 2N4401 NPN transistor in a TO-92 package. This allows the five volt low current I/O output from the microcontroller to control external items such as a vibration motor with ease. There is no current limiting on this connection, so a short on this external alarm output would blow the transistor. A simple current-limiting circuit could be added with a few more components.

One method that would probably be fast enough would be to insert an inline current sense resistor to the output and use this as feedback to an I/O pin that is capable of being interrupt controlled. Select the resistor value such that the current you are wanting to trip the over-





current at will be high enough to be seen as a digital high. The interrupt routine would simply turn off the output transistor's base current to shut things down. Nothing was done with this, though, to keep things nice and simple.

The Arduino Pro Mini has a reset pin on the printed circuit board (PCB). This was tied into a momentary N/O pushbutton switch that can be pressed to reset the system. This is used after the system has detected something with one of the three sensors and is now indicating which sensor caused the alarm using one of the three LEDs. The RST line on the board needs to be pulled to ground briefly to reset the microcontroller. The physical connection to this option was very convenient since the RST and GND pins are side by side.

Firmware

The code is very straightforward for this project. The hardest thing in this project is in the multiple ultrasonic sensor readings. However, this was made simple by using an Arduino library called NewPing by Tim Eckel (**http://playground.arduino.cc/Code/NewPing**).

A setup routine configures the hardware such that the correct pins are inputs and outputs. The ultrasonic sensors are then pinged one after another in the main loop. Once a new set of measurement data has been collected, the *pingDataComplete* function allows us to determine if any of the sensors fall within the alarm zone. If one does, we sound the alarm to scare off the cat and turn on the indication LED which corresponds to the sensor that caused the alarm. If all of the readings are normal, we just move on and take another round of readings.

During the alarm phase, we move the servo motor back and forth. Normally, you would code a PWM out to the servo so that it moved to the desired position. Another Arduino library has made this very simple. The servo library (**http://arduino.cc/en/reference/servo**) lets you take full control of the servo with just a few lines of code. Programming with some of these powerful libraries feels a bit like cheating if you are used to doing things the hard way, but I will take all the help I can get to speed up project building!

Building the Proto Board

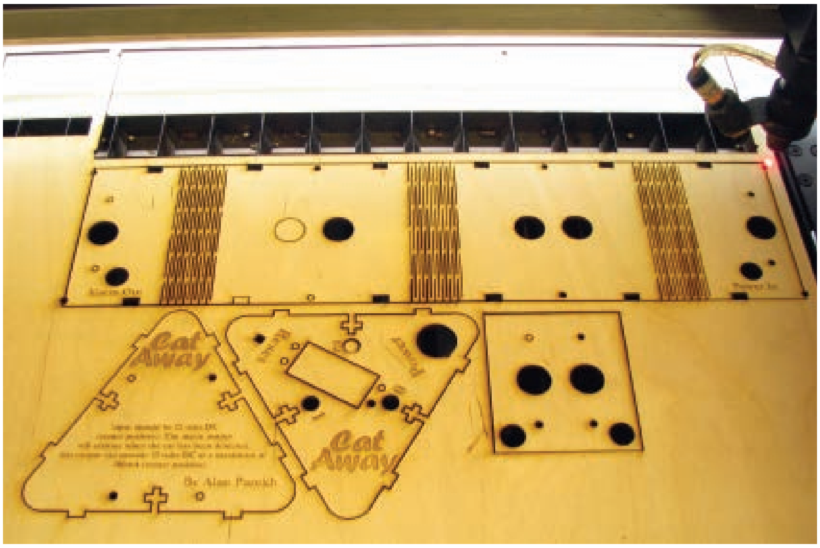
I struggled with the decision to breadboard the circuit first or just move to a perfboard since it was so simple. I have been bit by that in the past, so I ended up breadboarding the circuit first to true everything out. When making the permanent perfboard version, 0.1 inch breakaway headers were used to allow the entire thing to be modular. Of course, this can easily let out the magic smoke if a cable is plugged in backwards or into the wrong place. However, the benefits are ease of assembly and any troubleshooting that may be necessary.

Point to point wiring was used to connect everything together on the back of the perfboard. Many of the headers were placed such that they just needed to be linked to adjacent pins, reducing the chances of any shorts from crossing wires.

Designing and Building the Housing

As mentioned before, the triangular housing requirement meant it would need to be custom. Thankfully, the days of cutting plywood using a table saw for project boxes is over. A small CNC machine and laser cutter are now options. I opted on using the laser cutter since it is more precise, less messy, and would create something that would be strong enough for this build.

Initially, I was thinking of using 3 mm black acrylic for the design. The intersections of the three faces would be linked together using some shallow finger joints (just to keep the edges aligned), and a top and bottom section



would use some T-slot connections that would allow nuts and bolts to hold everything together. I could just imagine the language I would need to use when assembling the thing, though. I am sure the three side pieces full of electronics would not behave while I was trying to align and screw down the top and bottom of the housing.

I remember reading some articles about a laser cutting plywood in such a way to allow it to bend. Google "laser cut plywood bendable" for more information on this technique. I had some 3 mm mahogany and some 3 mm baltic birch plywood, so I gave the process a try. The results with the mahogany were not great. It seems inherently stiffer than the birch. After two or three design attempts, I had the birch bending like a limp noodle. The box design ended up being a long 39 cm strip that had three bendable joints cut into it so that it could be formed into a three-sided box. The back side of the box comes together in the center and is held tight using a small plywood brace and four nuts and bolts. The only holes that are cut into the long strip (other than 10 small bolt holes) that makes the walls are six holes for the three ultrasonic sensors to poke out of, and two holes for the power input jack and alarm output jack. The HC-SR04 ultrasonic sensors have four very tiny 1.4 mm mounting holes in the corner of the module. I didn't have any nuts and bolts this small, so I opted to simply use double-sided foam tape to hold them in place. This tape is crazy sticky, so it won't release unless you are deliberately trying to remove a sensor.

I think if I was going to spend a bit more time on the design, I would have built a support bracket that would hold the sensor board from the rear. The front has a crystal oscillator near the top, and the back is full of surface-mount electronics along with a 90 degree angled header, so foam tape would still be required as a spacer. The outer sticky portion could be left covered so it would be easier to work with, and then the sensor would not be able to be pushed in even with lots of force.

It took quite a bit of trial and error to get the perfect radius and triangle size for the upper and lower panels. After the first two or three didn't quite fit, I changed from plywood to cardboard until it finally came out, matching the curves of the bent corners exactly. The top and bottom

PARTS LIST

ITEM	DESCRIPTION	QTY	MODEL #/NOTES
U1	Five volt regulator	1	LM7805
U2	Arduino Pro Mini	1	Five volt 16 MHz version
For U2	24-pin wide DIP socket	1	Optional if Arduino is to be socketed
SW1	On/Off switch	1	
SW2	Momentary N/O pushbutton	1	
C1, C2	47 µF 25 volt capacitor	2	
C3	0.1 µF 50 volt capacitor	1	Any voltage rating over 10 volts is fine
R1 - R3	1/4 watt 470 ohm resistor	3	
R4	1/4 watt 1K ohm resistor	1	
Q1	NPN transistor	1	2N4401
D1 - D3	5 mm blue LED	3	
For D1 - D3	5 mm LED bezel	3	
M1	Servo motor	1	Dynam B2232
SEN1 - SEN3	Ultrasonic distance sensor	3	HC-SR04
Circuit Board	5 cm x 7 cm perfboard	1	

■ **Find tons of extra photos of this project at the article link.**

are each held in place with three T connections; once tightened, they hold everything in place very well. I am not sure how well it would hold up if I dropped it off the table, but it has a nice solid feel in the hand. The laser machine has the capability of etching onto the wood. This was used to etch in some identification information, as well as the project name. Getting the etch power just right so that the results are nice and dark takes some playing around, but thankfully baltic birch plywood is quite consistent.

There is actually quite a large number of items mounted to the top of this box. We have the on/off switch, a reset switch, three LEDs, a servo motor, and three slots for the T-slot nuts and bolts. I wanted the servo horn to be centered in the design which dictated where a lot of the other items needed to go.

The servo motor is held in place very securely using four nuts and bolts; the power switch and LED bezels are a simple press-fit design. The reset button was the only item on the top of the box that would have been better if I had a panel mount style momentary switch. The switch that I used is meant to be a replacement switch for computer reset buttons. These switches are snapped into a housing and have a button cap that presses it. It is mounted to the underside of the top panel using some hot glue, and has the switch shaft poking up through the top about 3 mm, which isn't too bad. Thankfully, the switch throw is only about 2 mm before it activates. The bottom panel doesn't have any holes cut into it other than four bolt holes for the perfboard to get mounted. Turns out a rectangular perfboard is not that simple to fit into a triangular base since there are mounting screws as well as the power input and alarm output jacks in close quarters. The perfboard is held off the bottom panel using 1/4 inch nylon spacers so that none of the solder connections are in contact with the case. It also allows for just enough space for the lower rear mounting screw to be inserted.

Testing It Out

I tried the project out with the servo disconnected so it wouldn't alarm the cats during testing. I wanted to see how well the ultrasonic sensors worked with a furry cat compared to the piece of cardboard I was using during testing. It was a bit of an afterthought that there might be a problem since I was just dealing with the sensors as magic boxes. However, in reality, they are sending out a burst of sound (above the human hearing range) and waiting for it to bounce off an object and return. One of our cats looks

like something you would see on the end of a boom mike (called a dead cat in the industry!) when they are being used outside to filter out wind noise. I was worried that the long haired cat might have a genetic way to bypass detection. Turns out the sensors work great in detecting both cats!

I set it up on the table without anything on the servo and waited to see what the reaction would be. With a few treats left on the table, the long haired cat (not the kitten) made a fast retreat when the servo started moving around.

When set up on the table, I was able to walk around the entire perimeter without being detected. There is a variable in the code called *tripDistance* which might need adjusting if your table needs a detection range greater than or less than the 40 cm that is default in the code.

Future Upgrade Possibilities

Some future changes that would enhance the functionality would be to mount a piezo buzzer or some other type of loud sounder directly in the box. One could be plugged into the alarm output jack, but it would be nice to have this functionality built in. Currently, there is only a single variable in code that sets the detection range of the sensors. This works great if it will just be used on a single table size that's round like my situation. A simple code change would allow for sensor 0, 1, and 2 to have different detection ranges. This would allow it to work better on oval tables, for example. This would introduce another obvious enhancement requirement, though. You would need to remember exactly where to place it so that all three sensors line up to the programmed sensor distances. None of these problems exist when using a nice round table.

The way to make that work the best would be to not only have three separate detection distance variables for the three sensors, but to also allow them to be programmed in place. This could be accomplished by adding a new switch to the system. When the switch is pressed, the system would make note of the closest distance it measures from all three sensors. You would simply walk around the entire table so that the system could record the safe distance. When put back into run mode, subtracting 5 cm from these new values would give you a great baseline for the system to have optimal detection.

Now that I've "tabled" this project, maybe you can think of some other useful applications. **NV**

By Dave Prochnow

Post comments on this article at **www.nutsvolts.com/index.php?/magazine/article/july2014_Prochnow.**

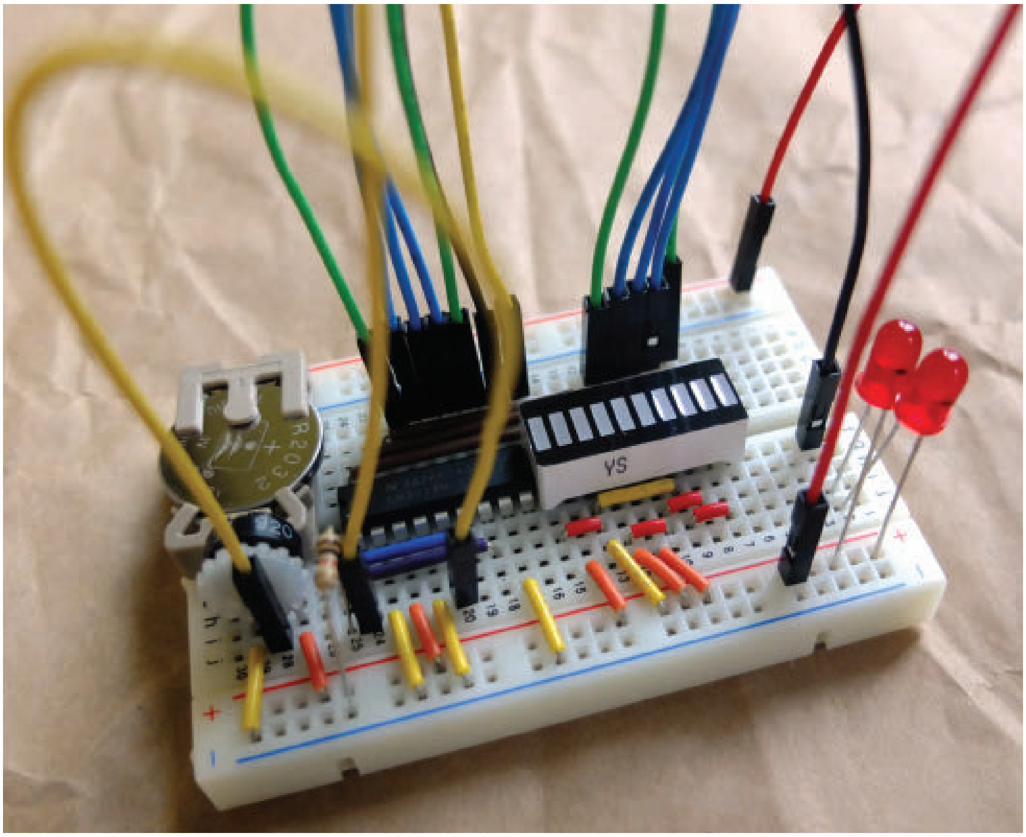
DESKTOP ROCKET FIREWORKS

Missed out on the Fourth of July fireworks this year? No problem! This inexpensive and easy-to-build "rocket" will help bring the excitement to your desktop. Or, if you did get to see a show, keep you entertained all year round. (Especially on those long days at work!)

Designed around the LM3914 dot/bar display driver IC and a 10-position yellow LED bar graph display, this fun little project operates on a single CR2032 3V lithium battery controlled by an SPDT switch. Therefore, you will be able to have a blast as often as you want without spending a fortune on batteries.

An oldie but goodie, the LM3914 is still being manufactured today and serves as the heart and soul of our desktop rocket.

You Can't Hold a Roman Candle to This



■ **Our desktop fireworks being tested on a breadboard. Note: The SPDT switch has not been added yet.**

Desktop fireworks is one of the simplest applications for the LM3914 as presented in the manufacturer's technical datasheet (NOTE: The Texas Instruments LM3914 technical datasheet was used during the development of this project), with just one resistor used in the circuit. Adding to this solitary resistor are two blinking red LEDs which will be used as replacements for the final LED in the bar graph (LED 10). In other words, rather than the LM3914 lighting LED 10 in the bar graph, the two blinking LEDs will be triggered. Figuratively, these blinking LEDs serve as the explosion of the aerial shell that typically culminates the launch of a rocket.

In order to gain the maximum impact with desktop fireworks, the entire circuit is mounted on a piece of lightweight corrugated cardboard stock and is cut into the shape of a rocket. Here is where you can get creative and let either whimsy or realism guide your rocket design. Regardless of your creative direction, however, remember to cut slots into the base of your rocket and on an extra set of fins. These two slots in the rocket and fins can then be slid together (forming a "+" base) for keeping your rocket on the "launchpad," enabling you to ignite a desktop fireworks display at a moment's notice.

After you've mounted the circuit on your cardboard rocket base and installed a 3V lithium battery, operation is as easy as 3-2-1, FIRE! First, flick the SPDT power switch; you've just lit the fuse. Slowly rotate the potentiometer; the rocket streaks skyward. When the top two LEDs remain lit (remember, the last LED — LED 10 — is NOT connected to the LM3914; instead the two blinking LEDs will act as the "last" LED), rotate the pot two more "clicks." The rocket explodes with bombs bursting in air. If you leave the final blinking LEDs powered, they will slowly ebb in and out of synchronization in a nice soothing visual effect.

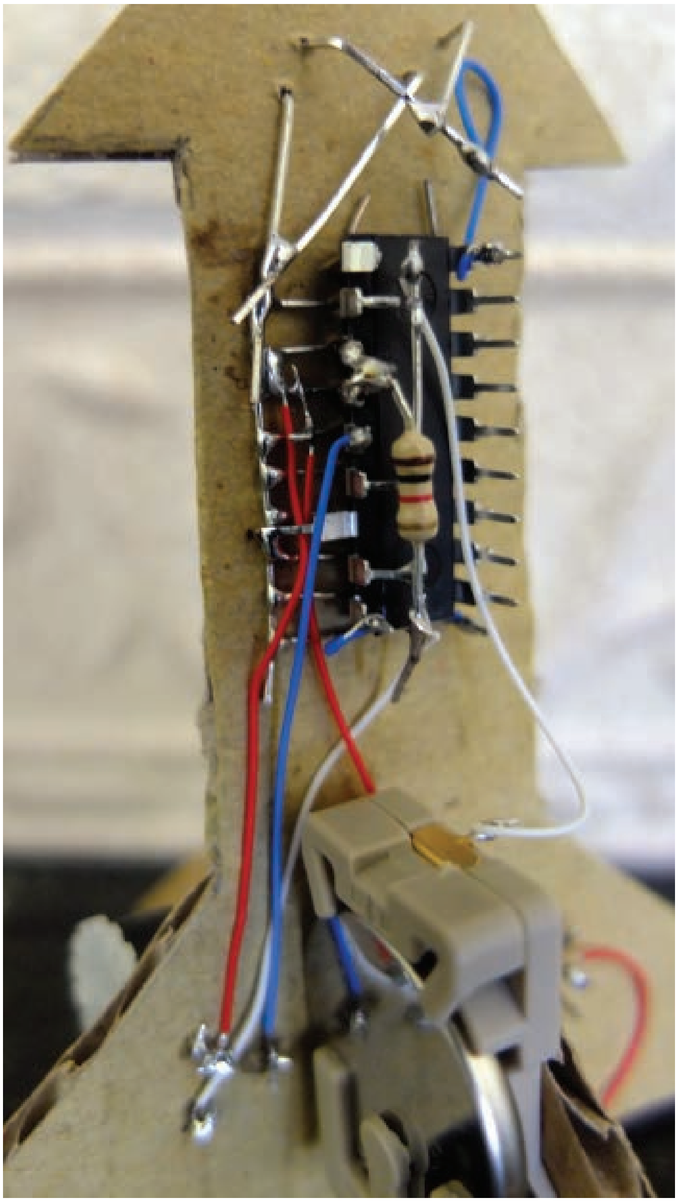
PARTS LIST

ITEM	COST SOURCE
LM3914	($1.95; SparkFun Electronics)
10-position LED bar graph	($1.95; SparkFun Electronics)
1K resistor	
(2) Blinking red LEDs	($.70 each; Mouser)
5K potentiometer	($1.50; Mouser)
SPDT switch	($1.50; SparkFun Electronics)
Coin cell battery holder	($1.25; SparkFun Electronics)
3V lithium battery	($1.95; SparkFun Electronics)
Hookup wire	

Prices subject to change without notice.

Step by Step

1. Push the LED bar graph's leads through the cardboard rocket until they protrude through the back.
2. Snip off the top, LED 10, cathode, and anode pins from the LED bar graph display.



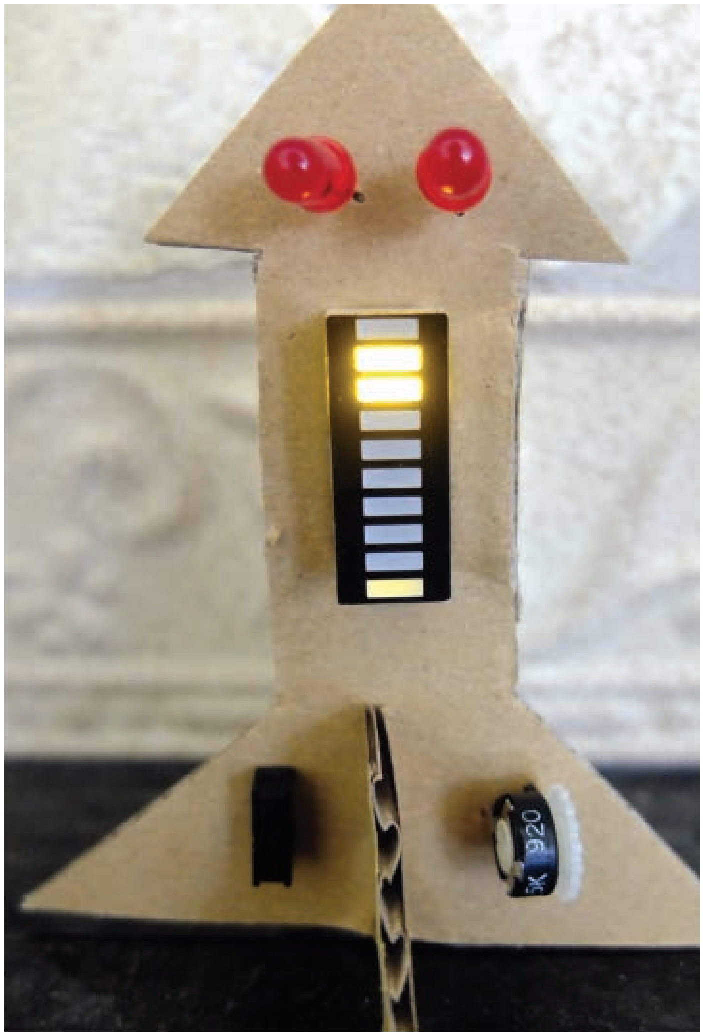
■ **The "business" end of the desktop rocket without lighting any fuses. Note: The LM3914 is fixed upside down and "dead bug" between the cathode/anode pin rows of the LED bar graph.**

3. Solder all of the remaining anode pins together. This anode pin bus will be connected to V+ during a later step.

4. Turn the LM3914 over *and* upside down (i.e., "dead bug"). Nest the IC neatly between the anode pin bus and the cathode pins.

5. Carefully solder pins 18-11 of the LM3914 to the adjacent cathode pins. These cathode pins are for LED 2 through LED 9.

6. Solder pin 1 on the LM3914 to the LED 1 cathode. Now, LEDs 1-9 on the LED bar graph are connected to the LM3914.



■ **Light up your life with desktop fireworks.**

7. Solder pins 6 and 7 on the LM3914 together, then connect a 1K resistor between these joined pins and ground (GND).

8. Connect the LM3914's pins 8, 4, and 2 to GND.

9. If you don't like the dot mode display for the desktop rocket, you can optionally connect pin 9 on the LM3914 to V+. This connection will enable bar mode display on the LED bar graph. In bar mode, all LEDs remain lit as the potentiometer is turned.

10. The signal source input is provided by the 5K potentiometer. The middle wiper contact on the potentiometer is connected to pin 5 on the LM3914. One of the two remaining potentiometer contacts is soldered to V+ and the final contact is soldered to GND.

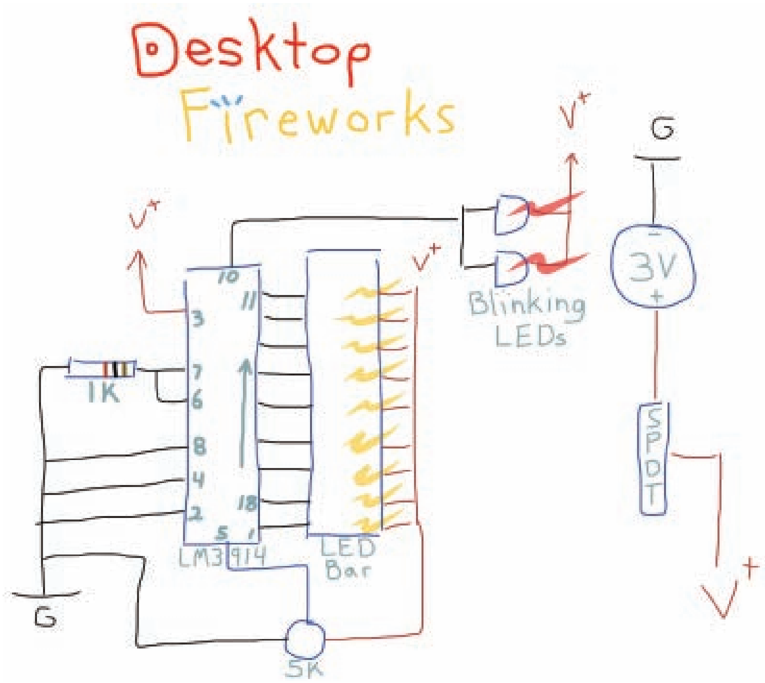
11. Carefully push the two blinking LEDs through the cardboard rocket's nosecone. The anodes of the LEDs are both connected to V+, whereas the cathodes of the blinking LEDs are connected to pin 10 of the LM3914.

12. The final step is to add the battery power. All GND connections are routed to the negative terminal (-)

of the battery holder, while the positive terminal (+) is soldered to the middle contact on the SPDT switch. The closed switch contact of the SPDT switch is then soldered to all of the circuit's V+ connections, including the anode pin bus that was created in Step 3.

Insert a 3V lithium coin cell into the battery holder and switch the circuit on. If none of the LEDs are illuminated, slowly rotate the potentiometer. When the pot is turned to one of its lower positions, the first LED (LED 1) will light up. In dot mode, one LED will light for each voltage increment increase throughout the travel of the potentiometer's wiper. Near the top of the LED bar graph, LEDs 8 and 9 will remain lit throughout the rest of the potentiometer's voltage range. When the equivalent of LED 10 is accessed with the pot, the two blinking LEDs will start flashing.

Now, all you have to do is add some whistle and explosion sound effects. At least you won't have to worry about being bugged by pesky mosquitoes, plus you can select your own musical accompaniment. That's a win-win. **NV**



■ **Wiring diagram.**

MakerPlot – The DIY Software Kit Part 10

By John Gavlik
and Martin Hebel

Post comments on this article at **www.nutsvolts.com/index.php?/magazine/article/july2014_MakerPlot**.

As we wrap up this particular series on how to customize MakerPlot, this time we're going to introduce you to how to create a custom Interface directly from your micro. The "normal" procedure for creating an Interface with meters, switches, pushbuttons, etc., is through macros or text instructions that are, in turn, created by the Object Editor on your PC; we discussed how this is done in the previous two articles. MakerPlot will accept the same set of text instructions via the serial port, just as it does analog and digital data from your micro. What this means is that your micro has full control over the MakerPlot Interface, as it can add and remove controls "on the fly" to suit the application.

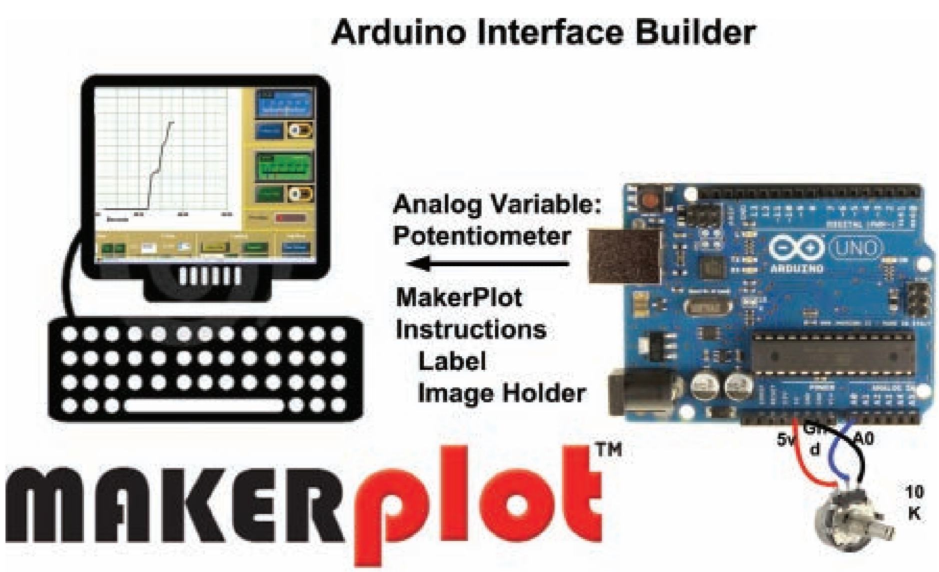


Figure 1. Arduino Interface Builder.

For example, if you're doing a process control operation and only want to monitor temperature, your micro can send just the MakerPlot instructions for displaying a meter to display temperature and nothing else. Then, if you want to switch to monitoring pressure, your micro can send the instructions to erase the temperature meter and put one up for pressure.

What we're saying is that you don't have to have everything on the MakerPlot Interface all at once; you can customize the "look" as needed right from your micro's serial port by adding or eliminating controls as necessary.

In this article, we'll show you an example of how to accomplish this feature by adding a couple of controls to monitor a potentiometer's analog value. If you have MakerPlot installed, you can follow along. If you haven't already done so, you can download a free 30 day trial copy of MakerPlot from **www.makerplot.com** to follow along. So, let's get going.

The Plan

Up to this point, we've created our Interface switches, buttons, text boxes, and so forth with macros using the Object Editor and Macro Builder (see Parts 8 and 9). Now, we're going to show you how to do it directly from your

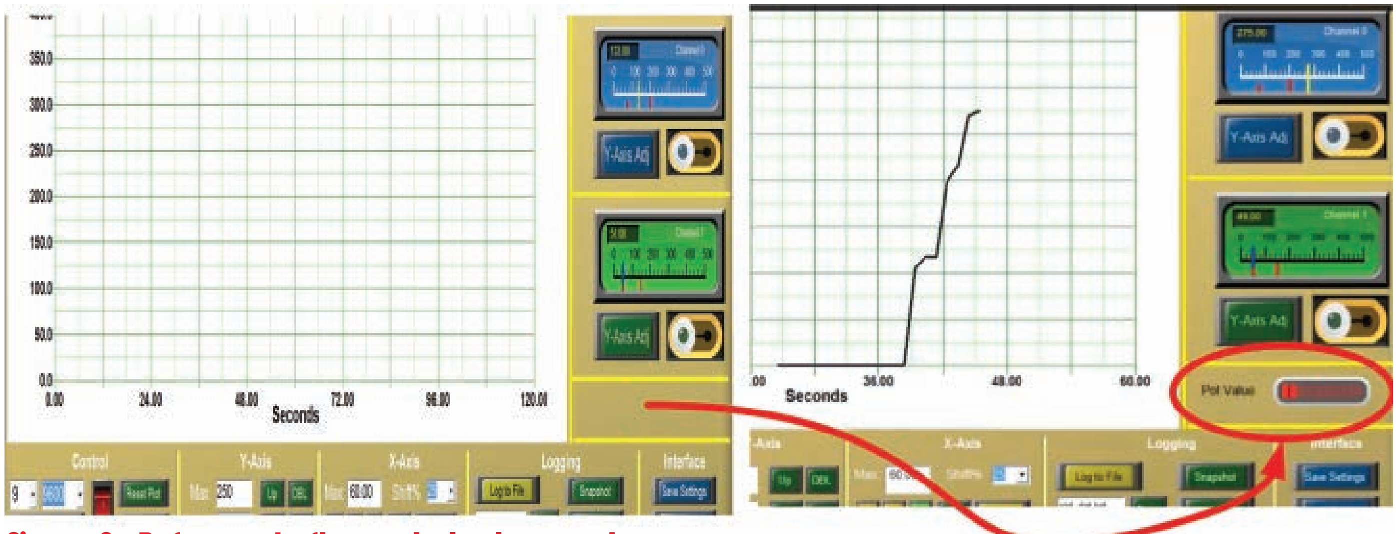


Figure 2. Before and after controls placement.

micro. With this capability, you'll be able to get to a new level of understanding of how MakerPlot works and, also, how you can create Interfaces that don't involve first loading a complete macro from the PC; the instructions will come directly from your micro — or at least a portion of them will in this example.

Figure 1 is where we're going to begin. This is our Arduino Uno with a 10K pot attached to the A0 analog input. To display the pot's value on MakerPlot, we're going to add two controls: a label called Pot Value and an LED image holder consisting of 10 LED bars to our (already customized) *My_Interface.spm* Interface. These two controls will NOT be on the Interface screen to begin with; rather, they will be added to it using code from the Arduino.

The Approach

So, let's discuss the best way to begin to place controls on MakerPlot directly from the Arduino. By the way, it doesn't have to be an Arduino; you can use most any of the popular micros that have serial output capabilities. The fundamental thing to realize is that all controls on the Interface have specific physical dimensions and placement coordinates.

Figure 2 shows the *My_Interface.spm* Interface with and without the controls we're referring to. It's in the marked rectangular area on the left where we're going to add them (as shown on the right). Once again, the goal is to know what the dimensions and placement coordinates are for these controls *before* you send the MakerPlot instruction via the micro's serial port. Here's how to do it.

The Technique

MakerPlot has an X,Y absolute coordinate system for placing controls in the Object Area; this is the area of the Interface that's not occupied by the Plot Area. The coordinates range from 0,0 at the bottom-left to 100,100 at the top-right. Use the *Values* Tab on the Toolbar to locate these X,Y coordinates as in **Figure 3**.

We've moved our cursor to the rectangular area where we want to place the controls that will come from the micro's serial port. We can then determine the exact

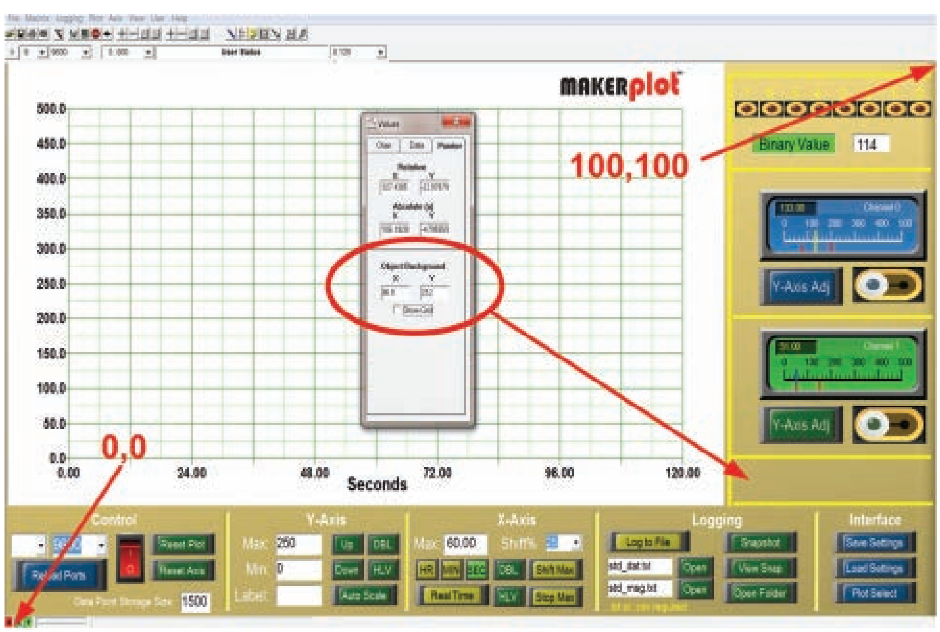


Figure 3. MakerPlot coordinate system.

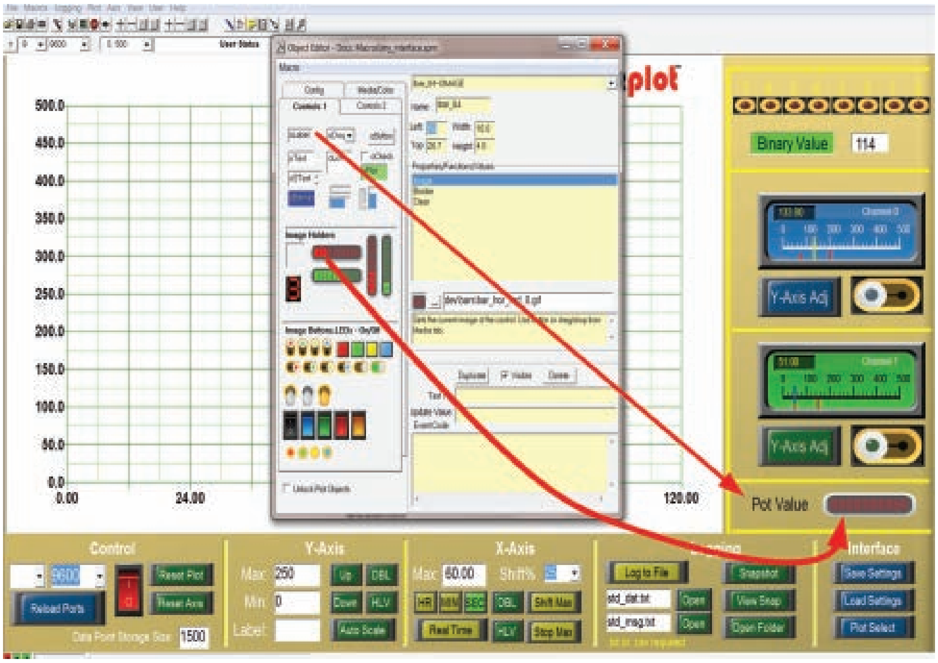


Figure 4. Manually placing controls.

manually place the controls on the Interface just as we did with the other controls — but not permanently. Then, we're going to let the Object Editor figure out all the pertinent control dimensions and coordinates for us.

Using the methods we showed you in Parts 8 and 9, in the rectangular area below the second meter we've placed a Label titled "*Pot Value*" and an LED bar graph Image Holder (**Figure 4**). These two controls were sized and oriented to fit exactly where we want them to go using the Object Editor.

Next, we brought up the Macro Builder (**Figure 5**) to see where these Objects are placed on the Interface. Clicking on the "*Update from current settings*" button, we can see that these objects appear as the last two in the Objects listing. This is what they look like:

location of the top-left of each control using the *Object Background* X, Y coordinates as shown. Then, we can calculate the sizes of each control and enter them into the instruction code for our micro.

While this is all doable, there's an easier way.

Rather than do all the calculations, we're going to first

```
`   lbl_83 — OLABEL *****
!POBJ oLabel.lbl_83=78.4,26.6,10.,3.,Pot
Value,7,0,8,0
!POBJ lbl_83.F=Arial,8,0,0,0
```

```
`   lbar_84 — OIMAGE *****
!POBJ oImage.lbar_84=88.,
26.7,10.,4.,dev\bars\bar_
hor_red_0.gif,0
```

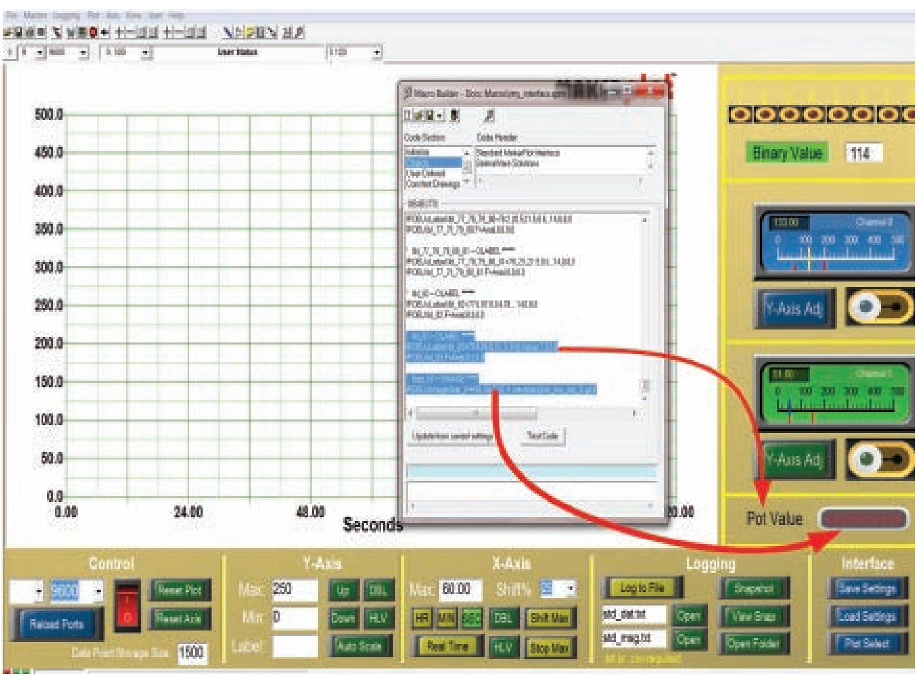


Figure 5. Copy the control instructions to the clipboard.

Notice that the Macro Builder has done all the work for us. It's included the MakerPlot *!POBJ* (Plot Object) instruction in front of the object types (*oLabel.lbl_83* and *oImage.lbar_84*), and then followed this with the X and Y coordinates and size dimensions — plus everything else that the object needs, like the label (*Pot Value*) and the type of LED bar graph (bars\bar_hor_red_0.gif). All of this is pretty cryptic, and while it can be done by hand without using the Object Editor and Macro Builder to configure the instructions, it's certainly more involved and risky compared to this method.

(The sidebar has an example of a handmade Interface to

illustrate that it can be done.)

The Sketch

All that's left to do is to "copy and paste" these instructions into our Arduino sketch. In doing so, you'll still need to prefix each line of text with the *Serial.println* instruction, as well as enclose each line of text in double quotes (" ") and end each line with a semi-colon to make it work. The final sketch should look like **Figure 6**. The sketch is fairly simple in terms of operation.

The Setup

After the potentiometer variables are declared, the setup configures the comm port to 9600 baud. Then, the pasted and edited lines are added as shown below:

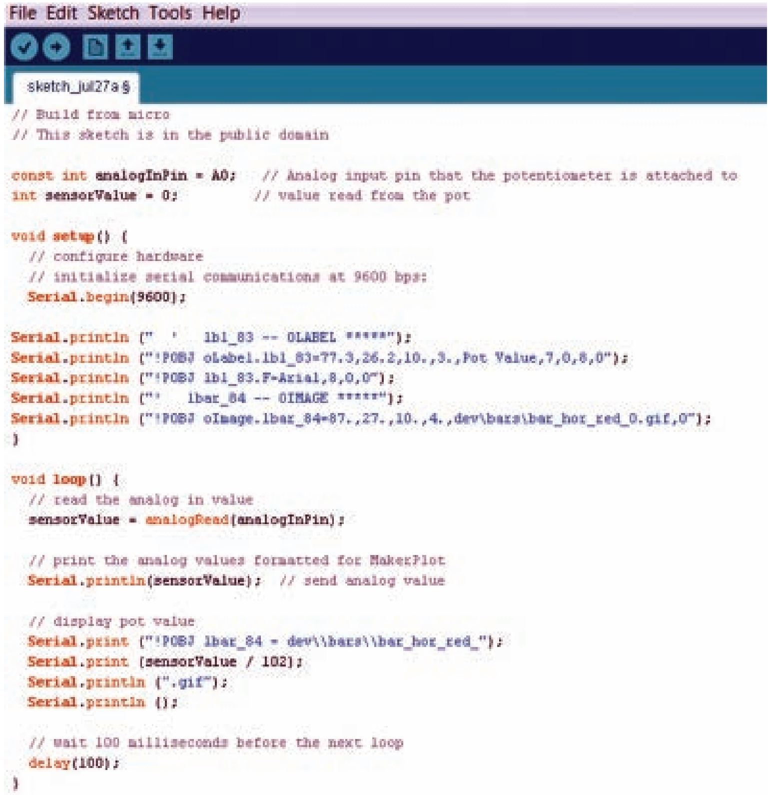
```
Serial.println ("  '   lbl_83 —
OLABEL *****");
Serial.println ("!POBJ oLabel.lbl_83=
77.3,26.2,10.,3.,Pot Value,7,0,8,0");
Serial.println ("!POBJ lbl_83.F=
Arial,8,0,0");
Serial.println ("'   lbar_84 — OIMAGE
*****");
Serial.println ("!POBJ oImage.
lbar_84=87.,27.,10.,4.,dev\bars\bar_hor_
red_0.gif,0");
```

Each instruction is on its own separate line. We don't really need the comments, but it's okay to leave them in for now. If we want to add more controls, it would be best to leave out the comments as it takes up microcontroller code space with something that MakerPlot doesn't really use. The take-away from the setup is that these instructions are sent to MakerPlot once to place the controls on the Interface. From here on, the loop takes over and updates both the potentiometer's analog value (a black plotting line), as well as the LED bar graph.

The Loop

Every 100 milliseconds, the raw 10-bit analog-to-digital (A2D) potentiometer value is read and sent to MakerPlot. That's handled by these instructions:

```
void loop() {
  // read the analog in value
  sensorValue = analogRead(analogInPin);
```




```
// Build from micro
// This sketch is in the public domain

const int analogInPin = A0;   // Analog input pin that the potentiometer is attached to
int sensorValue = 0;          // value read from the pot

void setup() {
  // configure hardware
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);

Serial.println ("  '   lbl_83 -- OLABEL *****");
Serial.println ("!POBJ oLabel.lbl_83=77.3,26.2,10.,3.,Pot Value,7,0,8,0");
Serial.println ("!POBJ lbl_83.F=Arial,8,0,0");
Serial.println ("'   lbar_84 -- OIMAGE *****");
Serial.println ("!POBJ oImage.lbar_84=87.,27.,10.,4.,dev\bars\bar_hor_red_0.gif,0");
}

void loop() {
  // read the analog in value
  sensorValue = analogRead(analogInPin);

  // print the analog values formatted for MakerPlot
  Serial.println(sensorValue);  // send analog value

  // display pot value
  Serial.print ("!POBJ lbar_84 = dev\\bars\\bar_hor_red_");
  Serial.print (sensorValue / 102);
  Serial.println (".gif");
  Serial.println ();

  // wait 100 milliseconds before the next loop
  delay(100);
}
```

Figure 6. Paste the instructions into the Arduino sketch.

```
  // print the analog values formatted for
  // MakerPlot
  Serial.println(sensorValue);
  // send analog value
```

The interesting part comes next; that is, how the LED bar graph gets updated.

The LED bar graph is nothing more than a group of 11 individual .gif images, starting with no LEDs illuminated to all 10 LEDs illuminated, and everything in between (**Figure 7**). As you can see, each image is labeled at the very end from 0 to 10, so it's up to the sketch to replace the image on the Interface each time a new potentiometer value is acquired and sent. It does this using the following code:

```
  // display pot value as an LED image bar
  // graph
  Serial.print ("!POBJ lbar_84 =
  dev\\bars\\bar_hor_red_");
  Serial.print (sensorValue / 102);
  Serial.println (".gif");
  Serial.println ();
```

Figure 7. LED bar graphs.

What this code snippet does is construct the name of the LED bar graph image based on the potentiometer value (or *sensorValue* in the sketch itself).

Here's how it's done:

```
Line 1: Serial.print ("!POBJ lbar_84 =
        dev\\bars\\bar_hor_red_");
```

The first line simply outputs a portion of the bar graph without the actual number appended at the end:

```
Line 2:  Serial.print (sensorValue / 102);
```

This line takes the *sensorValue* that can range from 0 to 1023, and divides it by 102 to create a "0 to 10" integer value result. This, then, corresponds to the number of the bar graph to send to MakerPlot:

```
Line 3:  Serial.println (".gif");
```

The next line appends the .gif suffix to the LED bar graph name; the *Serial.println* instruction also sends a carriage return, thus terminating the MakerPlot instructions. Note that Line 1 and Line 2 used the *Serial.print* that doesn't apply a carriage return:

```
Line 4:  Serial.println ();
```

The last line simply sends a blank carriage return that MakerPlot interprets as the final end of the previous instruction. The final result is the LED bar graph with the appropriate number of lit segments. Since we've already placed the first LED bar graph in the setup, all MakerPlot needs is the image name we give it from these four lines.

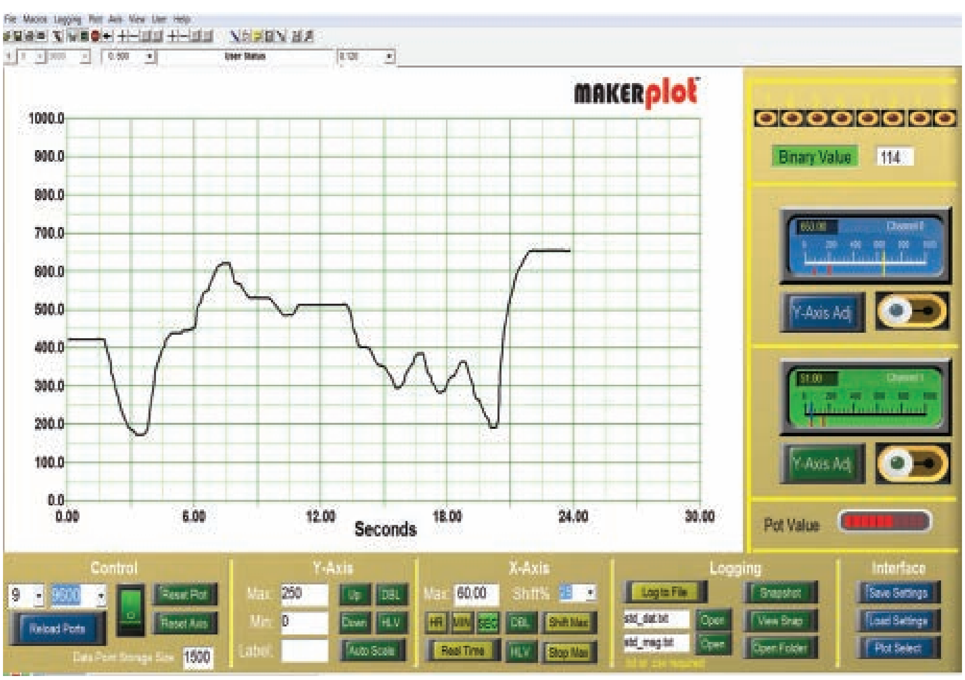


Figure 8. LED bar graph in operation.

The Sketch in Action

First, we need to run MakerPlot with the *My_Interface.spm* Interface. When the red rocker switch is clicked and the sketch is run for the first time, the two controls are placed in the rectangular area as in **Figure 4.**

As the potentiometer is adjusted, the LED bar graph illuminates with the correct number of bars as in **Figure 8**.

Conclusion

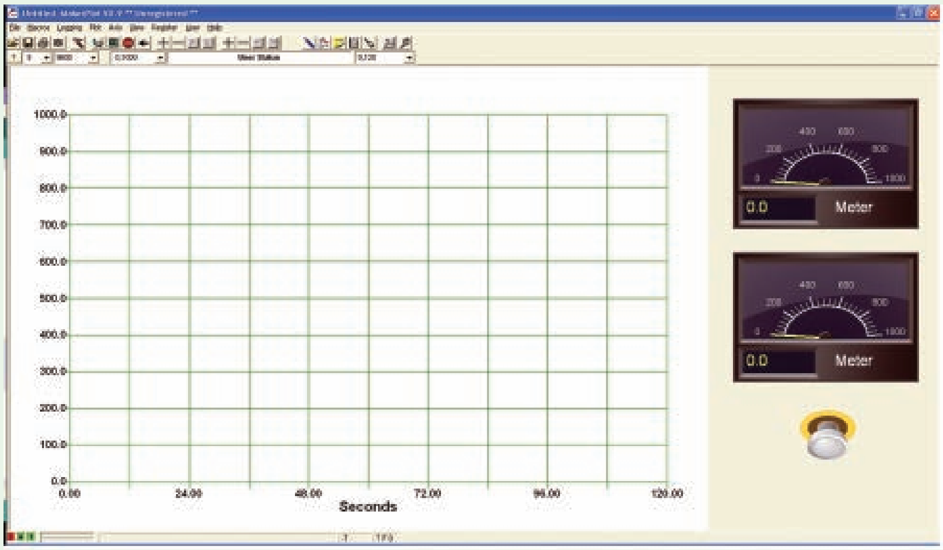
While this was a simple example of how to send controls to MakerPlot directly from a micro, it can be expanded to any number

of controls and other commands. You don't even need to start with an existing Interface like we did; your micro can handle all of the instructions to build, reset, and configure it. The sidebar shows an example of this. The ability for on the fly Interface building is unique to MakerPlot (as far as we know), and we've only shown just a few examples here. So, feel free to experiment with your application for a customized Interface builder using your favorite microcontroller.

We hope that you will continue your journey with this powerful software. There's plenty of info on the MakerPlot website to keep you going for quite a while. Remember, if you decide after using your 30 day free trial you want to purchase it, you can order it through the *Nuts & Volts* webstore (**http://store.nutsvolts.com/home.php?xid=a2d2eb6dd2786e0eb3c058e2aceda572**) at a discounted price.

That's all for now, so just remember: ***Got Data – MakerPlot It!*** **NV**

The following is an excerpt from the *MakerPlot Guide* that illustrates another example of creating an entire Interface from an Arduino microcontroller. The *Guide* can be found at **www.makerplot.com**. This text is accompanied by a graphic of the completed Interface. Note that the entire Interface is built, configured, and initialized from the Arduino — nothing was built on the PC at all. It's short and sweet, and it gets the point across.



Creating an Interface entirely from an Arduino.

Creating an Interface from the Arduino

One really interesting aspect of MakerPlot is that all instructions can come from a variety of sources for total control. This allows a controller to totally configure MakerPlot once connected — first, have the controller set up the interface. Then, have it size the plot and add meters and other controls for monitoring or interactive control.

The easiest way to do this is to design the Interface normally, build the Interface, then create code from the initializations and object code. Be sure to replace any / with //:

```
void MP_Config()
{
Serial.println("!NEWP");
       // Start a new plot
Serial.println("!SPAN 0,1000");
       // Span Y axis
Serial.println("!POBJ Clear");
       // Clear any objects
Serial.println("!PPER 75,100");
       // Size plot
// Create meters called 'met1' and 'met2' scaled
// 0 - 1000, auto update on channels 0 and 1.
Serial.println("!POBJ MP_METER_OBJ.met1=78.,93.,
20.,28.,0, 1000,Meter,Blue-Gray,0,1");
Serial.println("!POBJ
MP_METER_OBJ.met2=78.,60.,20.,28.,0,
1000,Meter,Blue-Gray,1,1");

// Create a toggle switch called 'toggle'.
// Add event code
Serial.println("!POBJ
oImgBut.toggle=85.,29.,6.5,14.,dev\\toggle-
sw\\gs_toggle_v_0.gif,0,dev\\toggle-
sw\\gs_toggle_v_1.gif,0");
Serial.println("!POBJ toggle.C='Play click
sound(;)");
Serial.println("~IWAV sw_toggle2.wav");
Serial.println("");
       // Extra CR to signal end of event code.
```

Once you connect, it is created and ready to meter two channels of analog data, plus allow the switch (toggle) to be read by your controller.

■ BY FRED EADY

LEARNING CORE VALUES

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/index.php?/magazine/article/july2014_DesignCycle**.

In the previous edition of Design Cycle, we chose an off-the-shelf Digilent hardware platform to support our EVE display project. We also flew an initial bomb run on the supporting XC32 firmware. During the time we've been apart, I've consumed quite a bit more of that elephant we referenced last time. This month, we will test and tune the basic functions that are necessary to enable the FT800 to drive an LCD panel.

■ Photo 1. To display this message and button, we had to initialize the FT800, establish some data structures, and issue commands via the MX3's SPI portal.

The Goal

Looking at **Photo 1**, it's rather obvious that we have already reached our initial goal. So, the focus of this month's discussion is how we got there. The key to our success lies in our ability to write an array of SPI functions that access the FT800 memory spaces.

The LCD panel is controlled using a low bandwidth SPI interface. In reality, the MX3's PIC32MX microcontroller is overkill as far as driving the LCD panel is concerned. The FT800 is designed to be driven using just about any microcontroller that can communicate over an SPI channel. Since FT800-generated images are rendered line by line, just about any low pin count and low memory resource microcontrollers can host the FT800.

Writing the Core Functions

When designing with microcontrollers, it is very important to understand how the registers and memory areas are organized. The same is true for the FT800 which is, in fact, a specialized microcontroller.

Right now, we are only interested in getting the display to function as we desire.

The FT800's 4 MB address space is also carved out to support touch and audio controller registers and memory buffers. You can get the details of the FT800 address

Start Address	End Address	Size	NAME	Description
0x000000	0x03FFFF	256 kB	RAM_G	Main graphics RAM
0x0C0000	0x0C0003	4 B	ROM_CHIPID	FT800 chip identification and revision information: Byte [0:1] Chip ID: "0800" Byte [2:3] Version ID: "0100"
0x0BB23C	0x0FFFFB	275 kB	ROM_FONT	Font table and bitmap
0x0FFFFC	0x0FFFFF	4 B	ROM_FONT_ADDR	Font table pointer address
0x100000	0x101FFF	8 kB	RAM_DL	Display List RAM
0x102000	0x1023FF	1 kB	RAM_PAL	Palette RAM
0x102400	0x10257F	380 B	REG_*	Registers
0x108000	0x108FFF	4 kB	RAM_CMD	Graphics Engine Command Buffer

■ Table 1. The actual firmware declarations that lay out these memory areas are physically coded within the *ft800-pic32mx.h* include file. Note that only the least significant 22 bits constitute a valid FT800 memory space address.

space by examining the FT800 memory map that is described in its datasheet.

All of the memory space we currently need to navigate is laid out in **Table 1**. Our job is to code a common set of calls that exercise the FT800 by reading and writing the registers and memory buffer areas defined in **Table 1**.

The FTDI documentation divides these reads and writes into three transaction types: Memory Read; Memory Write; and Command Write. The associated transaction type commands are Host Memory Read, Host Memory Write, and Host Command Write.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SPI Activity	Set SPI_SS_N Low (active)							
Command/Address	0	0	A21	A20	A19	A18	A17	A16
Address	A15	A14	A13	A12	A11	A10	A9	A8
Address	A7	A6	A5	A4	A3	A2	A1	A0
Dummy	x	x	x	X	x	x	x	x
Byte 0	Read Byte 0, MSB first							
...	Read Bytes..., MSB first							
Byte n	Read Byte n, MSB first							
SPI Activity	Set SPI_SS_N High (inactive)							

■ Table 2. A 24-bit address is passed physically, but the FT800 only utilized the least significant 22 address bits. This is a typical SPI read with an extended address field.

Host Memory Read

Table 2 is the top to bottom description of a Host Memory Read command. The FT800 acts as an SPI slave device and requires the toggling of the *SPI_SS_N* signal. We define the *SPI_SS_N* signal in the following manner:

```
#define CSlo                    LATGCLR = 0x0200;
   //0000 0010 0000 0000
#define CShi                    LATGSET = 0x0200;
```

The PIC32MX atomic operations are self-commenting. We have declared the FTDI documented *SPI_SS_N* signal as *CS*. According to **Table 2**, we must drop the *CS* line logically low to begin a Host Memory Read command. This is accomplished using *CSlo*. From the code, our *CS* signal is generated on RG9.

The Host Memory Read command ends when *CS* is forced logically high with the issuance of *CShi*. In the meantime, the Host Memory Read command is no more than a standard SPI read operation; the difference from a standard SPI read being the extended 22-bit address field.

We now have a basis for our Host Memory Read command code. Let's start by coding a single byte read command:

```
unsigned char rd8(unsigned int addr)
{
     unsigned char i;
     //convert address to Little Endian format
     addrBuf[0] = 0x3F & (addr >> 16);
     addrBuf[1] = addr >> 8;
     addrBuf[2] = addr;
     addrBuf[3] = 0x00;      //dummy byte

     CSlo;
     dummybite = SPI2BUF;    //clear BF
     for(i=0;i<4;++i)
     {
       SPI2BUF = addrBuf[i]; //send address
       while(SPI2STATbits.SPIRBF == 0);
          //wait until receive buff full
       dummybite = SPI2BUF;  //clear SPIRBF
       while(SPI2STATbits.SPIBUSY == 1);
          //all done?
     }

     SPI2BUF = 0x00;  //8 clocks out - 8 bits in
     while(SPI2STATbits.SPIRBF == 0);
       //wait until receive buff full
     readBuf[0] = SPI2BUF;   //read byte
     while(SPI2STATbits.SPIBUSY == 1);
       //all done?

     CShi;
     return(readBuf[0]);
}
```

The first order of business is to adhere to the FT800's rule concerning data format. So, we must first convert the 24-bit address to Little Endian format. As you can see in the code, we simply rotated the address information into an array in such a manner as to transmit the address top to bottom instead of bottom to top.

To save some code, the dummy byte is coded as part of the Little Endian array. *CS* pin RG9 is driven logically low and the SPI buffer full flag is cleared with a dummy read of the SPI data buffer. As per **Table 2**, 24 bits of address information is transferred to the FT800, followed by the dummy byte of 0x00.

SPI data travels on clock edges, so we need to clock the FT800 eight times to enable the PIC32MX on the MX3 to clock in one byte of information. Once we have filled the Master device's SPI buffer, we end the Host Memory Read command by forcing port pin RG9 logically high. The clocked-in SPI data resides in the first slot of the *readBuf* array. We simply return the byte of data as a byproduct of the Host Memory Read function call.

Another look at **Table 2** reveals that multiple bytes can be read with a single Host Memory Read command. Here is how we do that for two bytes:

```
unsigned short rd16(unsigned int addr)
{
     unsigned char i;
     unsigned short rdata16;
```

■ Screenshot 1. This is an MPLAB X debug window shot of the results of executing the *rd32* function against the FT800's *REG_FREQUENCY* register.

```
        //convert address to Little Endian format
      addrBuf[0] = 0x3F & (addr >> 16);
      addrBuf[1] = addr >> 8;
      addrBuf[2] = addr;
      addrBuf[3] = 0x00;      //dummy byte

      CSlo;
      dummybite = SPI2BUF;    //clear BF
      for(i=0;i<4;++i)
      {
           SPI2BUF = addrBuf[i];
             //send address
           while(SPI2STATbits.SPIRBF == 0);
             //wait until receive buff full
           dummybite = SPI2BUF;
            //clear SPIRBF
           while(SPI2STATbits.SPIBUSY == 1);
            //all done?
      }
      for(i=0;i<2;++i)
      {
           SPI2BUF = 0x00;
             //8 clocks out - 8 bits in
           while(SPI2STATbits.SPIRBF == 0);
             //wait until receive buff full
           readBuf[i] = SPI2BUF;      //read byte
           while(SPI2STATbits.SPIBUSY == 1);
              //all done?
      }
      CShi;
      rdata16 = make16(readBuf[1],readBuf[0]);
      return(rdata16);
}
```

Note that we are not simply reading two bytes from the FT800. We are actually reading a 16-bit integer value. The pair of bytes obtained from the FT800 is converted to an integer using the *make16* macro:

```
#define make16(varhigh,varlow) (((unsigned
short)varhigh & 0xFF)* 0x100) \
            + ((unsigned short)varlow & 0x00FF)
```

Another look at the position of the variables in the *make16* macro reveals that the *make16* macro is being used to convert the incoming Little Endian formatted data to Big Endian format.

There will be times that we'll need to transfer a 32-bit value from the FT800. The mechanics are exactly the same as the eight-bit and 16-bit reads we have previously discussed. With that, the 32-bit Host Memory Read command code looks like this:

```
unsigned int rd32(unsigned int addr)
{
     unsigned char i;
     unsigned int rdata32;
         //convert address to Little Endian format
     addrBuf[0] = 0x3F & (addr >> 16);
     addrBuf[1] = addr >> 8;
     addrBuf[2] = addr;
     addrBuf[3] = 0x00;      //dummy byte

     CSlo;
     dummybyte = SPI2BUF;    //clear BF
     for(i=0;i<4;++i)
     {
          SPI2BUF = addrBuf[i];
              //send address
          while(SPI2STATbits.SPIRBF == 0);
              //wait until receive buff full
          dummybyte = SPI2BUF;     //clear SPIRBF
          while(SPI2STATbits.SPIBUSY == 1);
              //all done?
     }
     for(i=0;i<4;++i)
     {
          SPI2BUF = 0x00;
         //8 clocks out - 8 bits in
          while(SPI2STATbits.SPIRBF == 0);
         //wait until receive buff full
          readBuf[i] = SPI2BUF;      //read byte
          while(SPI2STATbits.SPIBUSY == 1);
         //all done?
     }
     CShi;
     rdata32 = make32(readBuf[3],readBuf[2],
     readBuf[1],readBuf[0]);
     return(rdata32);
}
```

Instead of two bytes, four bytes are transferred from the FT800. The *make32* macro merges the four bytes into a 32-bit integer:

```
#define make32(var1,var2,var3,var4) \
    ((unsigned int)var1<<24)+((unsigned
    int)var2<<16)+ \
    ((unsigned int)var3<<8)+((unsigned
    int)var4)
```

Again, we use the *make32* macro to convert the received Little Endian value to Big Endian. Let's read the 32-bit *REG_FREQUENCY* register which holds the default Big Endian value of 0x02DC6C00. As you can see in **Screenshot 1**, the *REG_FREQUENCY* data came to the PIC32MX SPI portal in the sequence of 0x00, 0x6C, 0xDC, 0x02. The *scratch32* variable value is the result of the execution of the *make32* macro within the *rd32* function. Now that we can read, let's learn to write.

Host Memory Write

Table 3 looks just like **Table 2** with the exception of Write versus Read, and a read/write bit. The FT800 does not use the two most significant address bits, as its memory space is limited to 4 MB. In that these bits are always part of the data transfer, it's only logical to put

these unused bits to good use. Let's examine our single byte Host Memory Write command code:

```
void wr8(unsigned int addr, unsigned char data8)
{
     unsigned char i;
     //convert address to Little Endian format
     addrBuf[0] = 0x80 | (addr >> 16);
     addrBuf[1] = addr >> 8;
     addrBuf[2] = addr;

     CSlo;
     dummybite = SPI2BUF;
        //clear BF
     for(i=0;i<3;++i)
     {
          SPI2BUF = addrBuf[i];
        //write byte to SSPBUF
        //register
          while(SPI2STATbits.SPIRBF
          == 0);
        //wait until bus cycle
        //completes
          dummybite = SPI2BUF;
          while(SPI2STATbits.
          SPIBUSY == 1);
     }
          SPI2BUF = data8;
        //write byte to SSPBUF
        //register
          while(SPI2STATbits.SPIRBF
          == 0);
        //wait until bus cycle completes
          dummybite = SPI2BUF;       //clear BF
          while(SPI2STATbits.SPIBUSY == 1);
     CShi;
}
```

Now that we have an extra bit to twiddle, the Little Endian conversion code has to account for the read/write

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SPI Activity	Set SPI_SS_N Low (active)							
Command/Address	1	0	A21	A20	A19	A18	A17	A16
Address	A15	A14	A13	A12	A11	A10	A9	A8
Address	A7	A6	A5	A4	A3	A2	A1	A0
Byte 0	Write Byte 0							
...	Write Bytes...							
Byte n	Write Byte n							
SPI Activity	Set SPI_SS_N High (inactive)							

■ Table 3. The good news is that if we can write successful *read* functions, we have a very good chance at getting our *write* functions to work, as well. The only differences are the direction of the data transfer and a read/write bit.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SPI Activity	Set SPI_SS_N Low (active)							
Command	0	1	C5	C4	C3	C2	C1	C0
Zero	0	0	0	0	0	0	0	0
Zero	0	0	0	0	0	0	0	0
SPI Activity	Set SPI_SS_N High (inactive)							

■Table 4. There's nothing much to say about this as its intent is obvious to the most casual observer.

command bit. The Host Memory Read command Little Endian code also took care of the read/write command bit by default. Since the read/write command bit is set for a write operation, we must logically OR the bit into our Little Endian transfer package. An FT800 SPI write operation does not require the transmission of a dummy byte. Thus, we can immediately begin transferring data following the transmission of the 22-bit target address. The PIC32MX makes SPI transmission easy since all we have to do is drop the byte we want to send into the SPI buffer. Once the byte is determined to be in the SPI buffer, the PIC32MX SPI engine automatically clocks it out to the FT800.

Like the read operations, we will sometimes be forced to send multiple bytes to the FT800. The most important requirement is to perform all of the byte transmissions within the *CSlo* and *CShi* demarcation points:

```
void wr16(unsigned int addr, unsigned short
data16)
{
    unsigned char i;
    //convert address to
    //Little Endian format
    addrBuf[0] = 0x80 |
    (addr >> 16);
    addrBuf[1] = addr >> 8;
    addrBuf[2] = addr;

    writeBuf[0] = data16;
    writeBuf[1] = data16
    >> 8;

    CSlo;
    dummybite = SPI2BUF;
    //clear BF
    for(i=0;i<3;++i)
    {
        SPI2BUF =
        addrBuf[i];
    //write byte to SSPBUF
    //register
        while(SPI2STATbits.SPIRBF == 0);
         //wait until bus cycle complete
        dummybite = SPI2BUF;
        while(SPI2STATbits.SPIBUSY == 1);
    }
    for(i=0;i<2;++i)
    {
        SPI2BUF = writeBuf[i]
     //write byte to
            //SSPBUF register
        while(SPI2STAT
        bits.SPIRBF == 0);
            //wait until bus
            //cycle complete
        dummybite =
        SPI2BUF;
        while(SPI2STATbits.
        SPIBUSY == 1);
    }
    CShi;
}
```

To send a 16-bit integer, we must break the integer down into bytes. We are effectively repeating the *wr8* function for each byte we transmit. The same axiom is true for transferring a 32-bit value. We must break down the 32 bits into four bytes before transmitting:

```
void wr32(unsigned int addr,
unsigned int data32)
{
    unsigned char i;
    //convert address to Little
    //Endian format
    addrBuf[0] = 0x80 | (addr >>
    16);
    addrBuf[1] = addr >> 8;
    addrBuf[2] = addr;

    writeBuf[0] = data32;
    writeBuf[1] = data32 >> 8;
    writeBuf[2] = data32 >> 16;
    writeBuf[3] = data32 >> 24;

    CSlo;
    dummybite = SPI2BUF;
    //clear BF
    for(i=0;i<3;++i)
    {
        SPI2BUF = addrBuf[i];
    //write byte to SSPBUF
    //register
        while(SPI2STATbits.SPIRBF
        == 0);
```

Command	Value (including bits 6 & 7)	Description
Power Modes		
ACTIVE	0x00	Switch from Standby/Sleep modes to active mode. Write three bytes of 00h to issue the ACTIVE command
STANDBY	0x41	Put FT800 core to standby mode. Clock gate off, PLL and Oscillator remain on (default).
SLEEP	0x42	Put FT800 core to sleep mode. Clock gate off, PLL and Oscillator off.
PWRDOWN	0x50	Switch off 1.2V internal regulator. Clock, PLL and Oscillator off.
Clock Switching		
CLKEXT	0x44	Enable PLL input from Crystal oscillator or external input clock.
CLK48M	0x62	Switch PLL output clock to 48MHz (default).
CLK36M	0x61	Switch PLL output clock to 36MHz.
Miscellaneous		
CORERST	0x68	Send reset pulse to FT800 core. All registers and state machines will be reset.

■Table 5. This is it as far as commands for the FT800 go. We can execute any of them using our *hostCmd* function.

```
          //wait until bus cycle complete
         dummybite = SPI2BUF;
         while(SPI2STATbits.
         SPIBUSY == 1);
      }
      for(i=0;i<4;++i)
      {
         SPI2BUF = writeBuf[i];
          //write byte to SSPBUF
               //register
         while(SPI2STATbits.
         SPIRBF == 0);
               //wait until bus cycle
               //complete
         dummybite = SPI2BUF;

while(SPI2STATbits.SPIBUSY
         == 1);
      }
      CShi;
}
```

Note that in all of the multi-byte transmission functions, the data is not converted to Little Endian before transmission. The FT800 receives the data as-is and stores it in Little Endian format.

We now have the power to read and write the entire FT800's 4 MB memory space. However, there is another function we need to add to our FT800 toolbox.

Host Command Write

The contents of **Table 4** are quite different than what we have encountered so far. Sending an FT800 command is a relatively simple process. Fire off a command byte followed by a couple of zero bytes:

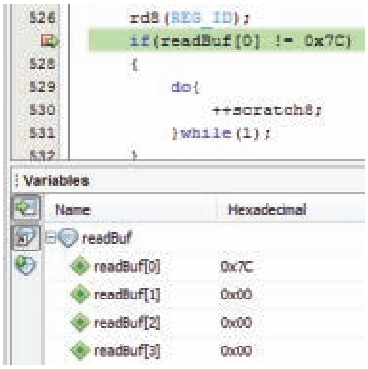
```
void hostCmd(unsigned char
cmd)
{
     unsigned char i;
     CSlo;
     dummybite = SPI2BUF;
        //clear BF
     SPI2BUF = cmd;
        //send command
     while(SPI2STATbits.SPIRBF
     == 0);
        //wait until bus cycle
        //complete
     dummybite = SPI2BUF;
     while(SPI2STATbits.
     SPIBUSY == 1);
     for(i=0;i<2;++i)
```

RESOURCES

Digilent, Inc.
chipKit MX3
www.digilentinc.com

FTDI
FT800
www.ftdichip.com

Microchip
XC32 C Compiler
MPLAB X
PICkit 3
www.microchip.com



■ Screenshot 2. Just in case you're not from Tennessee, high cotton means tall cotton — and we seem to be walking in it.

```
    {
        SPI2BUF = 0x00;
          //send 0x00
        while(SPI2STATbits.
        SPIRBF == 0);
          //wait until bus
          //cycle complete
        dummybite = SPI2BUF;
        while(SPI2STATbits.
        SPIBUSY == 1);
    }
    CShi;
}
```

Table 5 lists the commands that we can transfer and execute using the *hostCmd* function.

SPI Setup

While I was chewing on a piece of elephant, I noticed that my original SPI setup was not working as designed. The problem seemed to be in the initial wake-up speed of the PIC32MX's SPI clock. I lowered the SPI clock speed from 10 MHz to 4 MHz, which solved the "dead display" problem. Here's the updated SPI configuration code that runs in the *init()* function:

```
SPI2CONbits.ON = 0;
   //disable SPI2 peripheral
SPI2CONbits.MSTEN = 1;     //Master mode
SPI2CONbits.CKP = 0;
   //polarity idle low
SPI2CONbits.CKE = 1;
   //transmit on active to idle
SPI2CONbits.SMP = 0;
   //sample in the middle
SPI2BRG = 0x13;
   //4MHz SPI clock
SPI2STATbits.SPIROV = 0;
   //clear overflow flag
SPI2CONbits.ON = 1;
   //enable SPI2 peripheral
```

Waking the FT800

Before we can access the FT800, we must wake it up. Naturally, before we can throw that bucket of cold water on it, we must execute the *init()* function to set up the PIC32MX's GPIO, clock, and SPI portal. The PIC32MX *init()* function also insures that the *CS_N* and *PD_N* (Power Down) pins are forced to their inactive states.

There are a couple of user-accessable LEDs on the MX3 that are initialized. If you are new to PIC32MX programming, one of the most important gotchas to look out for is the JTAG interface which is enabled by default:

```
#define CSlo         LATGCLR = 0x0200;
        //0000 0010 0000 0000
#define CShi         LATGSET = 0x0200;

#define PDlo         LATBCLR = 0x0020;
     //0000 0000 0010 0000
#define PDhi     LATBSET = 0x0020;

#define LD4on    LATFSET = 0x0001;
    //0000 0000 0000 0001
#define LD4off   LATFCLR = 0x0001;
#define LD4tog   LATFINV = 0x0001;
#define LD5on    LATFSET = 0x0002;
    //0000 0000 0000 0010
#define LD5off   LATFCLR = 0x0002;

    CShi; //SPI slave select inactive
    PDhi;  //power down inactive
    LD4off;  //utility LEDs on MX3
    LD5off;
    DDPCONbits.JTAGEN = 0;
          //disable JTAG interface
```

I included the lo-hi-on-off mini macros for clarity. Using the PIC32MX atomic I/O operations eliminates the need for a schematic just to identify these particular I/O pin assignments. The FT800 wake-up sequence begins by driving the FT800's *PD* pin logically low for 20 mS, and then driving the *PD* pin logically high for 20 mS:

```
  init();     //execute initialize
```

Register	Description	WQVGA 480 x 272	QVGA 320 x 240
REG_PCLK_POL	Pixel Clock Polarity	1	0
REG_HSIZE	Image width in pixels	480	320
REG_HCYCLE	Total number of clocks per line	548	408
REG_HOFFSET	Horizontal image start (pixels from left)	43	70
REG_HSYNC0	Start of HSYNC pulse (falling edge)	0	0
REG_HSYNC1	End of HSYNC pulse (rising edge)	41	10
REG_VSIZE	Image height in pixels	272	240
REG_VCYCLE	Total number of lines per screen	292	263
REG_VOFFSET	Vertical image start (lines from top)	12	13
REG_VSYNC0	Start of VSYNC pulse (falling edge)	0	0
REG_VSYNC1	End of VSYNC pulse (rising edge)	10	2

■ Table 6. I figured I had nothing to lose, so I plugged these values into the driver. As it turned out, I was able to continue walking in the cotton field.

```
          //function
PDlo;     //drive PD low
delayms(20);     //wait 20mS
PDhi;            //drive PD high
delayms(20);
          //wait 20mS
hostCmd(ACTIVE);
          //write 0x00 0x00 0x00
hostCmd(OSCEXT);
          //write 0x44 0x00 0x00
hostCmd(PLL48);
          //write 0x62 0x00 0x00
rd8(REG_ID);
          //read addr 0x102400
if(readBuf[0] != 0x7C)
          //did read return 0x7C?
{
    do{
        ++scratch8;
    }while(1);
}
```

After the *PD* pin toggle operation, the *hostCmd* function is used to send the *ACTIVE* command, which is essentially three transmissions of 0x00.

Table 5 calls it *CLKEXT*, but the definition in the *ft800-pic32mx.h* include file declares it *OSCEXT*. The *OSCEXT* command enables the FT800 to recognize an external crystal or clock source:

```
//Host Commands Contained Within ft800-pic32mx.h
#define OSCINT       0x48
#define OSCEXT       0x44
#define PLL48        0x62
#define PLL36        0x61
#define PLL24        0x64
#define ACTIVE       0x00
#define STANDBY      0x41
#define SLEEP        0x42
#define PWRDN        0x50
#define CORERST      0x68
```

The next step is to set the FT800 internal clock speed to 48 MHz using the *PLL48* command. At this point, we can optionally increase the speed of the SPI clock. In that we're learning to crawl, I've chosen to leave the SPI clock alone at this time. We should now be able to read the *REG_ID* register. If the read returns a 0x7C, we are walking in high cotton. Check out **Screenshot 2**. The FT800 is almost ready to go to work. Although an LCD panel is present and attached, the FT800 does not yet have enough information to use it. So, we'll give the FT800 the particulars it needs to drive the WQVGA LCD panel:

```
wr16(REG_HCYCLE, 0x0224);
wr16(REG_HOFFSET, 43);
wr16(REG_HSYNC0, 0);
wr16(REG_HSYNC1, 41);
wr16(REG_VCYCLE, 292);
wr16(REG_VOFFSET, 12);
wr16(REG_VSYNC0, 0);
wr16(REG_VSYNC1, 10);
wr8(REG_SWIZZLE, 0);
wr8(REG_PCLK_POL, 1);
wr8(REG_CSPREAD, 1);
wr16(REG_HSIZE, 480);
wr16(REG_VSIZE, 272);
```

The aforementioned LCD panel settings can be obtained from the LCD panel's datasheet. Typical values are shown in **Table 6**. As you can see, we used our homegrown *wr8* and *wr16* functions to load the LCD panel data.

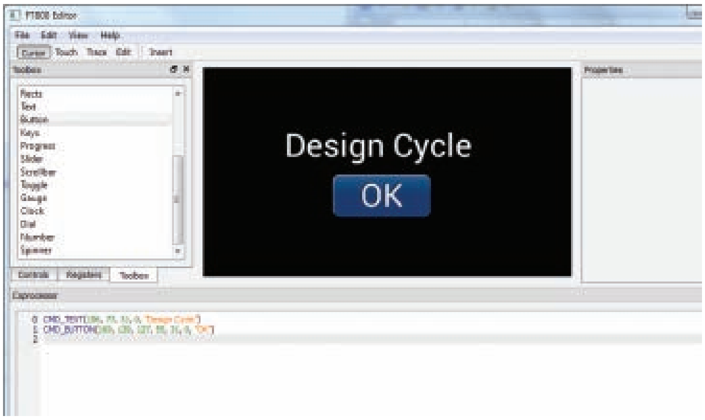
Are We There Yet??

Yes, but we're out of our allotted word space. So, I'll leave you with the code that produced the graphic details you saw back in **Photo 1**:

```
cmdBufRd = rd32(REG_CMD_READ);
cmdBufWr = rd32(REG_CMD_WRITE);
cmdOffset = cmdBufWr;

cmd32(CMD_DLSTART);
cmd32(CLEAR_COLOR_RGB(0,0,0));
cmd32(CLEAR(1,1,1));

sprintf(txtBuf,"Design Cycle");
drawText(106, 77, 31, 0,txtBuf);

sprintf(btnMsg,"OK");
drawButton(169, 139, 127, 55, 31, 0,btnMsg);

cmd32(DISPLAY());
cmd32(CMD_SWAP);
wr32(REG_CMD_WRITE,cmdOffset);

wr8(REG_GPIO_DIR,(rd8(REG_GPIO_DIR)) | 0x80);
wr8(REG_GPIO,(rd8(REG_GPIO_DIR)) | 0x80);
wr8(REG_PCLK,5);
```

There are a few functions included in the display code sequence that we still need to discuss. We will examine the yet to be identified code I've presented in detail next month. In the meantime, you may want to download the FT800 Editor from the FTDI site. The FT800 Editor will help you get a tighter grip on the ways of the FT800 without having to have any physical EVE hardware.

With that, I'll leave you with **Screenshot 3**. **NV**



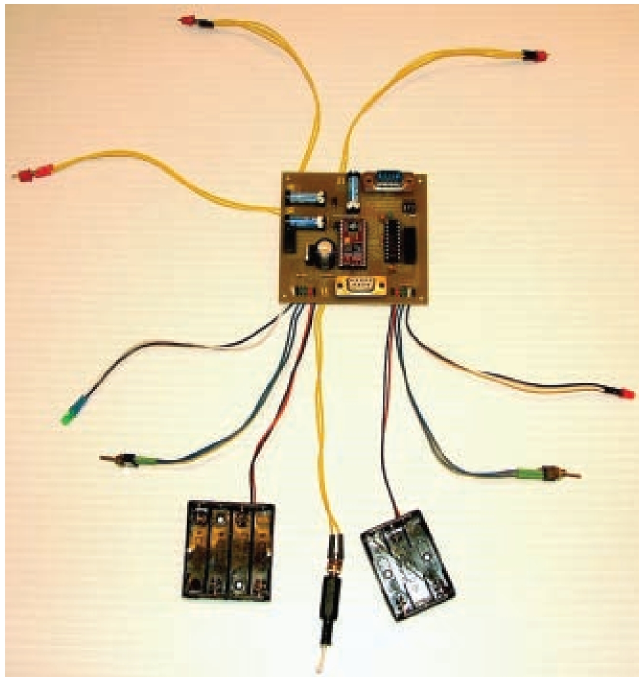
■ Screenshot 3. If you want to get a better handle on the FT800 coordinate system and command set, download this tool. It does not require actual EVE hardware and runs on your PC.

BY L. PAUL VERHAGE

The BalloonSat Extreme

Post comments on this article and find any associated files and/or downloads at **www.nutsvolts.com/index.php?/magazine/article/july2014_NearSpace**.

Sometimes a simple flight computer just isn't enough for all your near space experiments. That's when a BASIC Stamp 2pe really comes in handy. This month, I'll describe my largest BalloonSat flight computer: the BalloonSat Extreme.



■ Nope, it's not a robotic octopus. This is a very large BalloonSat flight computer built around a BASIC Stamp. When one BalloonSat flight computer isn't enough, this is your baby.

The BalloonSat Extreme's BS2pe is powerful enough to operate experiments for multiple students. With the Extreme, I see a classroom of student teams designing individual experiments and then integrating them into a single large BalloonSat. This design and integration process simulates the design and construction of real spacecraft like the New Horizons.

New Horizons

New Horizons is a mission to explore Pluto and the Kuiper Belt. The little spacecraft has so much kinetic energy that it will cross the solar system in only 10 years. As a result of its large kinetic energy, it will sail past this "Mickey Mouse" of a world and right out of the solar system. More importantly, there are seven experiments onboard to characterize Pluto and the outer solar system.

Scientists/researchers responsible for each experiment assembled them into a single New Horizons spacecraft. One neat example is its student designed and built dust collector experiment. This instrument will show how the concentration of dust varies across the entire soar system.

The point I'm making with New Horizons is that it wasn't designed and built by a single team. Large missions of exploration like New Horizons are the product of lots of scientists, researchers, and engineers coming together to produce the most capable vehicle of exploration that their budget allows. Teachers can replicate the same model of design

and exploration using BalloonSats if the flight computer is large enough.

An Extreme BalloonSat Flight Computer

As mentioned, the core of the BalloonSat Extreme is Parallax's BASIC Stamp 2pe. The BS2pe has 16 memory banks of 2 kb each for a total of 32 kb of memory. So, there's lots of room for programs. Since BASIC is the programming language of the BS2pe, the learning curve for new students is pretty shallow.

The power of Parallax's version of BASIC means that in most cases, only a single 2 kb memory bank is sufficient to operate a near space mission. This leaves the remaining 30 kb of memory available for data storage. That's enough memory for 15 separate experiments.

However, a flight computer can't collect and store data unless it can interface with a wide variety of sensors. So the BalloonSat Extreme flight computer is designed with five external ports.

First is the flight computer's analog port. The BS2pe doesn't have built-in analog-to-digital conversion capability, meaning it can't process sensor voltages directly into digital values. Therefore, it relies on a MAX186 IC to provide this service.

This ADC is pretty slick. It has eight channels and digitizes voltages with 12 bits of resolution (for a total of 1024 voltage bins). Its maximum voltage is capped to 4.096 volts, so with those 12 bits of resolution, it's capable of measuring voltage changes as small as one millivolt.

To make interfacing analog sensors easier, each channel of the analog port provides a connection to +5V, ground, and one channel of the MAX186. All it takes to operate an analog sensor is to plug it into one channel of this port.

The second external interface is the servo port. This port has three male headers that connect servos using the standard Futaba connector. The BalloonSat Extreme uses a separate battery pack for the servos. The reason why is a separate battery pack prevents a stalled servo from draining the main battery and crashing the flight computer due to a low voltage brown-out.

Next is the camera port where three of the BS2pe's I/O pins connect to relays. These relays take the place of a camera's shutter switch, and allow the flight computer to trigger up to three cameras to take a picture. With three cameras, a BalloonSat can simultaneously record images in all directions.

The fourth port is the digital port. Five BS2pe I/O pins are assigned to this one and like the analog port, each channel in the digital port provides ground, +5 volts, and an I/O connection to the BASIC Stamp. The digital port can operate experiments like a Geiger counter — a sensor that produces a five volt pulse every time a cosmic ray passes through.

The last port is the GPS port. This port is a male DB-9 connector firmly bolted and soldered to the printed circuit board (PCB). It connects to one I/O pin so the BS2pe can collect data like mission time and altitude. The port uses one of the DB-9's pins for serial data, pin 4 for positive five volts, and pin 5 for the ground connection. Because of the connection to power, a GPS can start providing position data as soon as it's plugged into the port.

There remains one more external connection: the commit pin. It's the method used to start the BS2pe program operating experiments and collecting data. That way, the BalloonSat Extreme doesn't record a significant amount of data on the ground.

Construction

There are three steps to assembling the BalloonSat Extreme: making the PCB; soldering components to the PCB; and wiring the flight computer. I'm fond of using DALPRO (now CamCon Chemical) products to make PCBs. However, you may be more familiar with other products. Whatever the method, use the bottom copper pattern available at the article link to make your PCB.

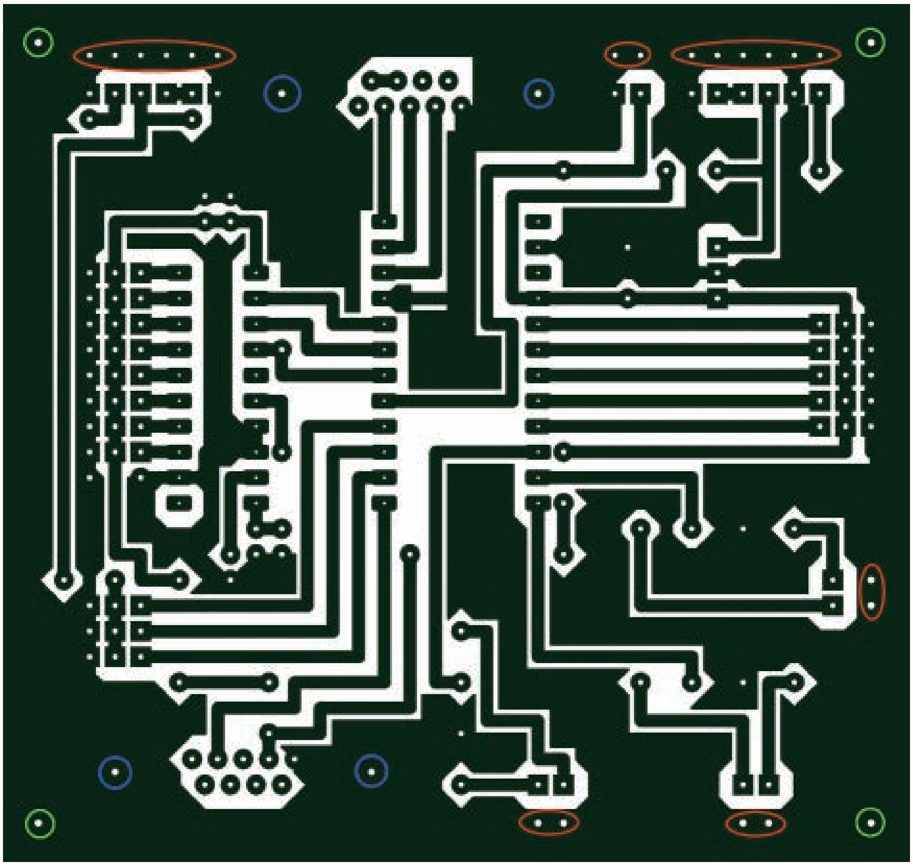
Drilling the PCB

The majority of the holes in the PCB are for component leads, but here are 28 holes you need to drill in a larger diameter.

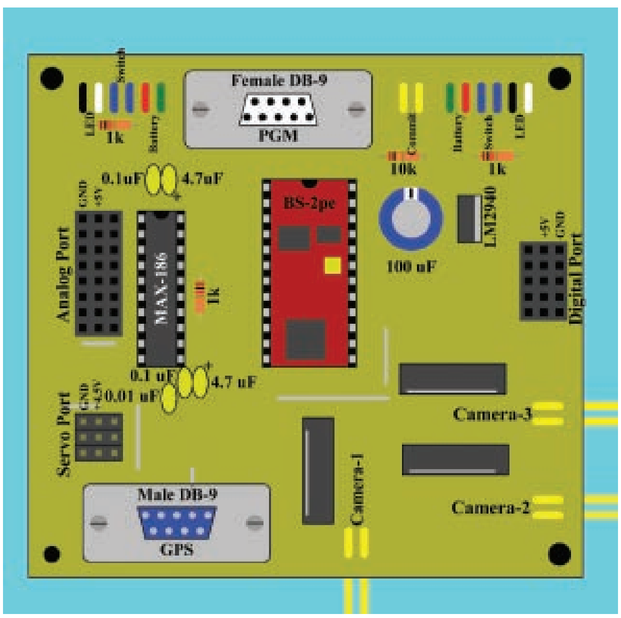
When I taught electronics in high school, I and my students hated

PARTS LIST

- Four AAA cell holder
- Three AAA cell holder
- Two sub-miniature toggle switches (use toggle switches with solder lugs)
- 24-pin IC socket 0.600 inches wide
- 20-pin IC socket 0.300 inches wide
- Male DB-9 header (PC mount)
- Female DB-9 header (PC mount)
- LM2940 five volt voltage regulator
- Three reed relays (use Jameco item #1860088)
- 100 µF or larger electrolytic capacitor (tantalum is preferable)
- Two 4.7 µF electrolytic capacitors (tantalum is preferable)
- Two 0.1 µF capacitors
- 0.01 µF capacitor
- Three 1K resistors
- 10K resistor
- 3x5 female receptacle
- 3x8 female receptacle
- 3x3 male header
- BASIC Stamp 2pe
- MAX186 12-bit ADC



■ **FIGURE 1.** The 20 strain relief holes are circled in green, the PCB mounting holes are circled in red, and the DB-9 mounting holes are circled in blue.



■ **FIGURE 2**. Follow this diagram for the placement of parts in your etched and drilled printed circuit board.

when a solder kit had wires soldered directly to the PCB with no other support. Invariably, the wires would soon break off from metal fatigue. I discovered a way to prevent this, and suspect many other people have discovered the same method.

I provide strain relief to the wires by drilling larger holes near the solder pads. Insulated wires pass through these larger holes before their stripped ends are soldered to the PCB. Now, the wires can flex without straining and breaking their soldered connection to the PCB.

There are 20 strain relief holes in the PCB for this purpose, and they are drilled to match the outside diameter of your insulated wire.

The remaining eight holes are mounting holes. The four holes in the corners of the PCB are where the flight computer is mounted to the base material. Drill these holes large enough for #4-40 bolts. The last four holes are for mounting the DB-9 connectors. Drill these holes for #2-56 bolts. Refer to **Figure 1**.

Soldering Components to the PCB

This is the order of assembly that I recommend for putting the PCB together (your mileage may vary). Start with the six jump wires (these are used to bridge traces beneath the PCB and are shown as the white lines in **Figure 2**), followed by the resistors.

Next, solder IC sockets. Do not solder the BASIC Stamp or the MAX-186 directly to the PCB, and do not insert the ICs into their socket prior to soldering the socket. Hold off inserting the BS2pe and MAX186 until after testing is complete.

You'll probably want to cut the analog and digital port receptacles from a longer block of receptacles. Do this by pulling out the row of pins you need to cut through and using a hack saw to do the cutting. Finish the receptacle by using a utility knife and file to smooth the roughly cut edge of the plastic body.

Insert one of the receptacles into the PCB and only solder two diagonal corner pins. Then, flip the PCB over and verify that the receptacle is still flush on the PCB. If it's not, press on the high corner and reheat that pin until the receptacle snaps down. Now, you can solder the receptacle's remaining pins. Repeat the process for the second receptacle.

Unlike the analog and digital ports, the servo port is a male header

(3x3 pin). Insert the short pins of the header into the PCB and solder it into place like you did for the digital and analog receptacles.

Before soldering the DB-9 connectors to the PCB, bolt them to the board. Use two 1/4" long spacers between the DB-9 and the PCB, and two 1/2" long #2-56 bolts. It's best to use a nylock (locking nylon nut) on the bolts. That way, the nuts can't vibrate or shake loose.

One last thing. Watch that the male DB-9 is used for the GPS port and the female DB-9 is used for the programming port.

What if you don't have a PC mounted DB-9 handy? You can use a DB-9 with solder cups if you solder short bare wires to the one. It's easiest to prep the DB-9 with a set of helping hands, as soldering the wires seems to be a three-handed job.

After stabilizing the DB-9, heat one solder cup with a soldering iron and dab a bit of solder inside the cup opening. Use just enough solder to fill the cup. Cut a short wire — like a resistor lead — and use pliers or tweezers to hold the wire against the solder-filled cup. Then, reheat the cup until the solder melts and the wire is soldered to the cup. Repeat this for the remaining eight solder cups.

Solder the three relays next. These have built-in protection diodes, so they are polarized devices. The white line drawn on the relays in **Figure 2** represents the pale white lettering on the side of the relay body.

Last, solder the LM2940 voltage regulator. It's also a polarized device and installed with its heatsink facing the right side of the board. The regulator doesn't sit flush on the PCB.

Now, the PCB is complete except for the wiring and testing. My next column will finish the BalloonSat Extreme by discussing wiring, testing, and using the flight computer in a BalloonSat.

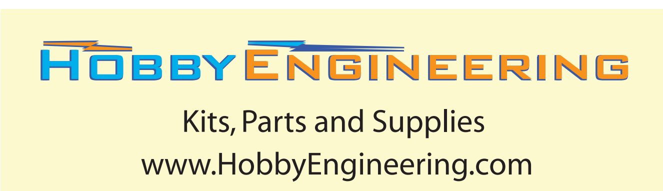
In the meantime, you'll want to become familiar with programming the BASIC Stamp 2pe and storing data in its memory. One unique feature of this BalloonSat flight computer is that it can record data from a GPS receiver. So, review how to read and parse serial data, as it's needed to collect data from the GPS receiver.

Speaking of GPS receivers, if you're looking for one to use in your BalloonSat Extreme or any other near space project, make sure it's a GPS that will report data above 60,000 feet. There are many models that stop producing position reports above this altitude and they are nearly useless for near space exploration.

Onwards and Upwards,
Your near space guide **NV**

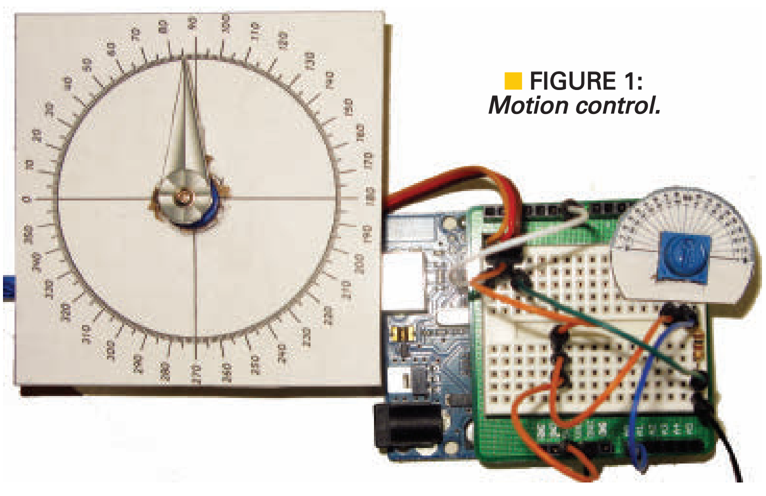
ELECTRONET

For the ElectroNet online, go to
www.nutsvolts.com
click ***Electro-Net***



The Arduino Classroom
Arduino 101 — Chapter 7: Analog Input — Part 2

Last month, we introduced concepts for reading analog data from the environment using the Arduino ADC (analog-to-digital converter). We also learned how to use software to provide a very simple way to use a PC to send commands and data to the Arduino and then receive replies from the Arduino. This is a great tool for using and debugging applications. We then got a mile-high overview of two of the core concepts of electricity: Ohm's law and circuits. We kept it simple and hopefully didn't lose anyone along the way. [If you felt lost, please ask questions on the www.arduinoclassroom.com forum.] In this chapter, we will learn a bit more about analog input using a device that lets us set a dial angle position to tell the computer what we want it to do. This device — a potentiometer — will also help reinforce our understanding of Ohm's law and circuits, while providing a very useful tool for further work in both computing and electronics. The capstone lab for this chapter will be to create an application that lets us turn a dial on a breadboard and have the new dial position be reflected by a servomotor pointer as shown in Figure 1. While this may seem like a simple thing to do (and it is), it's also one of the fundamental tools for folks controlling electronic and computer systems.



■ FIGURE 1: *Motion control.*

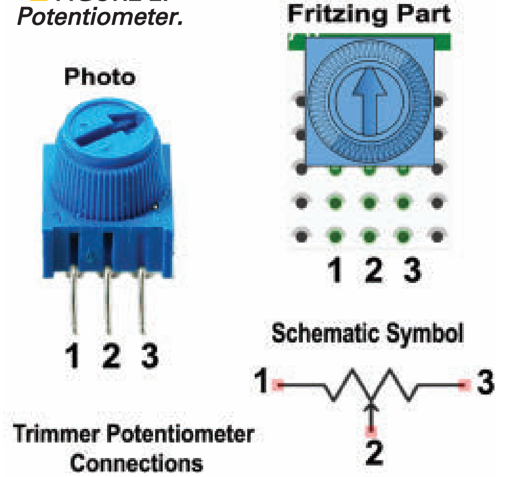
Variable Resistance: The Potentiometer

A potentiometer (or pot) is a mechanical device that contains a strip of resistive material connected on each end to pins, with a third pin in the middle called a wiper. The wiper can be made to slide across the resistive material such that the resistance varies depending on the wiper's position. In effect, it is a continuously variable voltage divider. **Figure 2** shows this concept in the schematic symbol. Pin 2 is the wiper; pins 1 and 3 are on either end.

When you turn the knob, the slider moves along the resistive material. The resistance between the slider pin and the end pins varies proportionally to the slider position between the two end pins.

If we apply a voltage as shown in **Figure 3** where we have five volts on the left pin and zero volts on the right pin, we then have a voltage divider as discussed in Chapter 6. The difference is that the voltage divider from last month was made with discrete 1,000 Ω resistors (each having a

■ FIGURE 2: *Potentiometer.*



Post comments on this article and find any associated files and/or downloads at
www.nutsvolts.com/index.php?/magazine/article/july2014_Smiley.

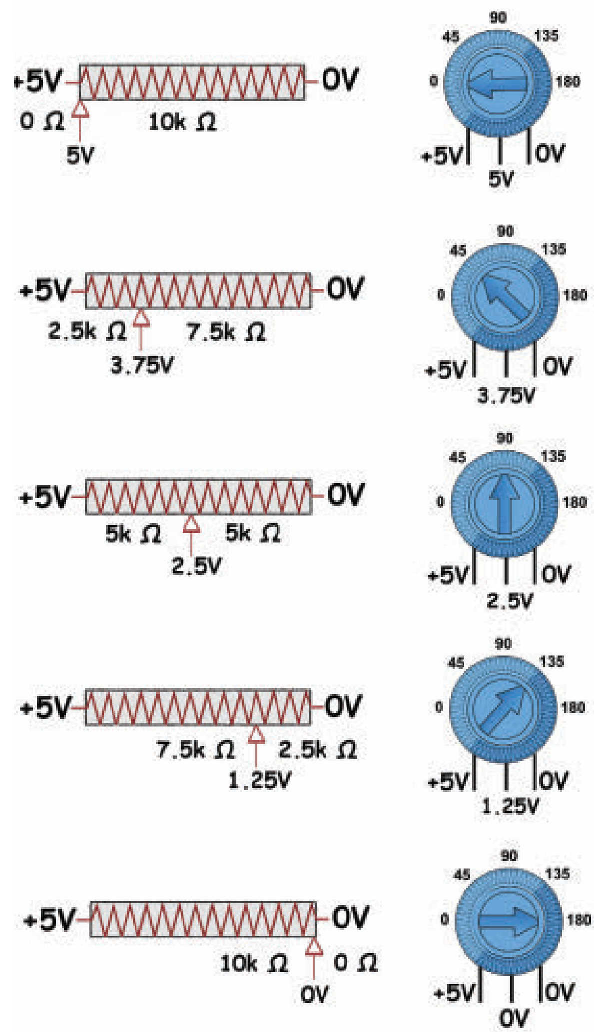


FIGURE 3: *The pot wiper and voltage.*

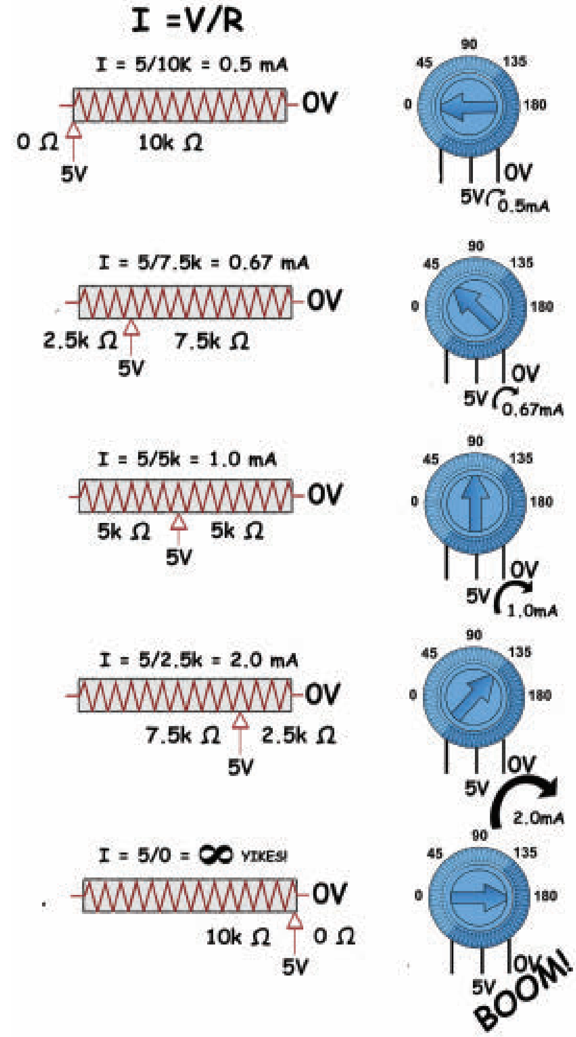


FIGURE 4: *The pot wiper and current.*

constant value), while the pot voltage divider resistance varies smoothly as the slider moves, with a portion of the resistance on one side and the remaining resistance on the other. With the discrete 1,000 Ω resistors, you could access the divide in 11 steps, from 0 Ω to 10,000 Ω. Each step above 0 is 10% of the total. However, in our case, you can move the slider to access a continuously variable resistance from 0 to 10,000 Ω.

The slider divides the total resistance such that the resistance on each side of the slider always adds up to 10,000 Ω. If the slider is exactly in the center, then there is 5,000 Ω to the left and 5,000 Ω to the right. If the slider is three-quarters of the way from the left to the right, then 7,500 Ω is to the left and 2,500 Ω is to the right as shown in the second to the last illustration in **Figure 3**.

Knob Position and Resistance

When we apply a voltage across the pot — five volts on the left and zero volts on the right as shown in **Figure 3** — then the slider pin will have a voltage proportional to the angle position of the knob. Again, if it is in the middle of the slider, the knob arrow will be pointed straight up at 90° (if we assume a half circle with 0 at the left and 180 at the right). If the position is one-quarter of the way from the left to the right (2,500 Ω to the left and 7,500 Ω to the right), then it is at the 45° position. If it is three-quarters of the way, then it is 135° (7,500 Ω to the left and 2,500 Ω to the right). This is also shown in **Figure 3**. We will use this relation between the angle and the voltage divider values to control the angle of our servomotor.

Two Modes of Operation: Voltage and Current

In **Figure 3**, we see the pot operated in the voltage mode where the slider position reflects the proportion of

the voltage divided between the left and right resistance. We would use this mode of operation when we want to manually set a voltage that is used in some other part of an electronic system. For instance, we might have the pot hooked up to a voltage controlled audio amplifier. We would then adjust the audio volume by moving the knob to the left and right while listening to the sound volume change until it was where we want it.

We might also want to communicate a magnitude to a computer. We could then have the wiper pin connected to an ADC that would measure the voltage and tell the computer what the voltage is. Then, the software would make decisions based on that voltage. An example of this would be to set the starting drill position in a CNC (Computerized Numeric Control) machine. You'd have one pot to set the drill's X position and a second pot to set the Y position. When the drill is right where you want it, you could press a button telling the computer to start the drilling. The computer would read the X and Y voltages and 'know' the location of the zero points for both the X and Y directions in the work surface.

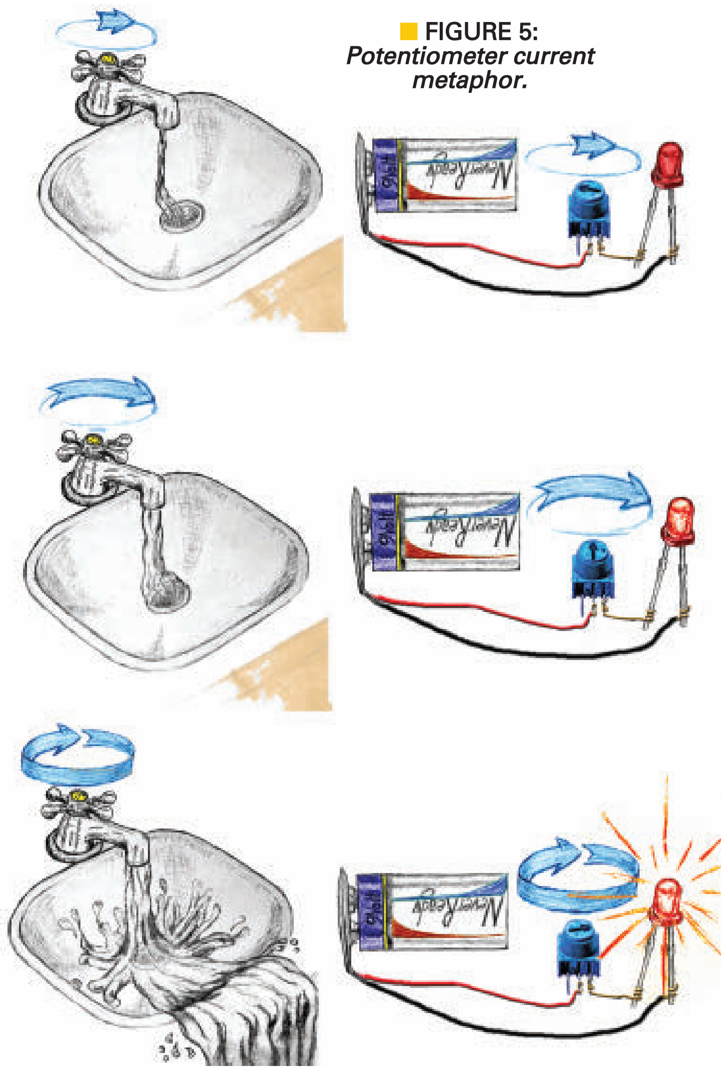
You can also use a pot to control an electric current as shown in **Figure 4**. If you only use the wiper pin and one of the other pins, you now have a variable resistor — not a resistor divider. If you apply the input voltage to the wiper pin, then as you move the wiper that input voltage is across a varying resistance (from 10,000 Ω when turned all the way to the left, and 0 when turned all the way to the right). Since we know by Ohm's law that the current is set by the voltage and resistance, we know that the current varies with the resistance which is set by the position of the wiper. This configuration is called a **rheostat** and is used to control current.

Figure 4 shows how this works. In the case of the wiper being all the way to the right (notice how it doesn't work ... boom!), you see a physical illustration of the old 'divide by zero' error. You can't divide by zero nor can you apply a voltage across zero ohms. You can try, but the system will attempt to generate an infinite current — which, of course, it can't.

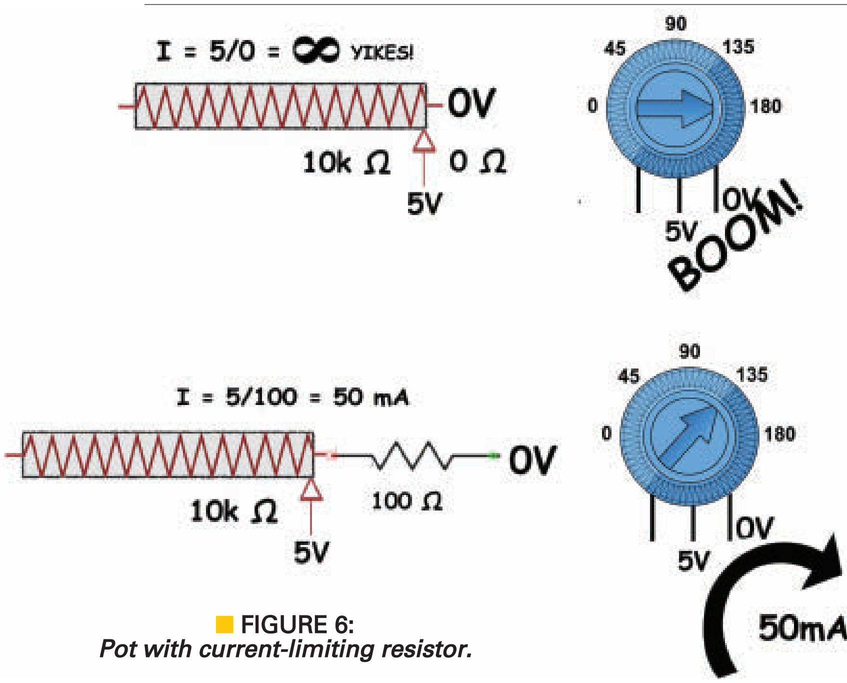
Figure 5 shows this concept with a water current metaphor and a soon-to-be burned out LED. To prevent this situation, you will want to add a resistor in series with the side of the pot being used in the rheostat mode.

Figure 6 shows a 100 Ω resistor used as a **current-limiting resistor**. Never forget to add this resistor since as we've already seen, applying any voltage across zero resistance is a sure path to disaster.

Let's get our hands dirty in some labs now.



■ FIGURE 5:
Potentiometer current metaphor.



■ FIGURE 6:
Pot with current-limiting resistor.

Lab 1: Potentiometer Analog Control for LED Brightness

Parts required:
1 Arduino
1 USB cable
1 Arduino proto shield
and jumper wires
1 LED
1 Potentiometer
1 100 Ω resistor

Estimated time for this lab:
15 minutes

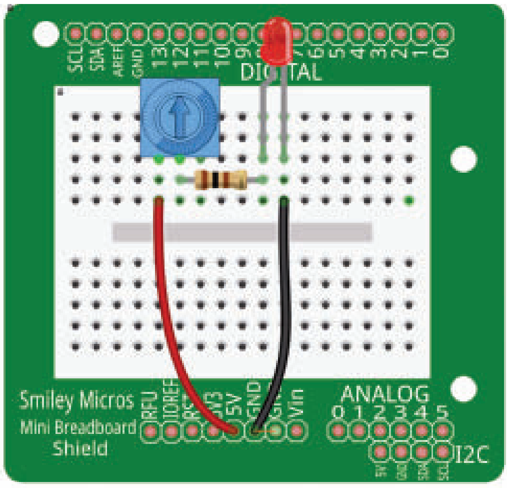


FIGURE 7: *Potentiometer LED breadboard drawing.*

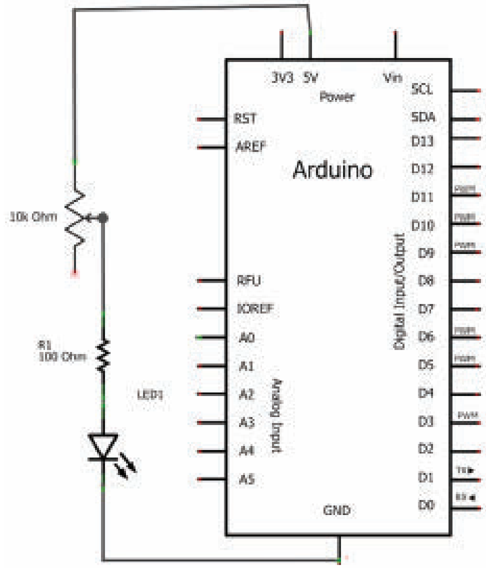


FIGURE 8: *Potentiometer LED breadboard schematic.*

Check off when complete:
❑ Build the circuit shown in **Figures 7** and **8**.
❑ Vary the pot angle and note the effect on the LED.
❑ Draw this circuit with arrows showing current flow.
❑ Calculate the current flow with the pot set at full right, three-quarters right, middle, three-quarters left, and full left.
❑ Calculate the voltage across the 100 Ω resistor at each of these currents.

Lab 2: Potentiometer Digital Voltage Control - ADC

In the last lab, we calculated the current with the pot set at full right, three-quarters right, middle, three-quarters left, and full left. We then calculated the voltage across the 100 Ω resistor for each of those positions. Now, we will verify those calculations by using the Arduino ADC to measure the voltage across the 100 Ω resistor.

Parts required:
1 Arduino
1 USB cable
1 Arduino proto shield
and jumper wires
1 LED
1 Potentiometer
1 100 Ω resistor

Estimated time for this lab: 15 minutes

Check off when complete:
❑ Build the circuit shown in **Figures 9**, **10**, and **11**.
❑ Copy the following program into the Arduino IDE (available at the article link):

```
// A101_ch7_pot_voltage 5/7/14 Joe Pardue

int sensorPin = A0;    // analog input pin
int sensorValue = 0;   // store the analog input
```

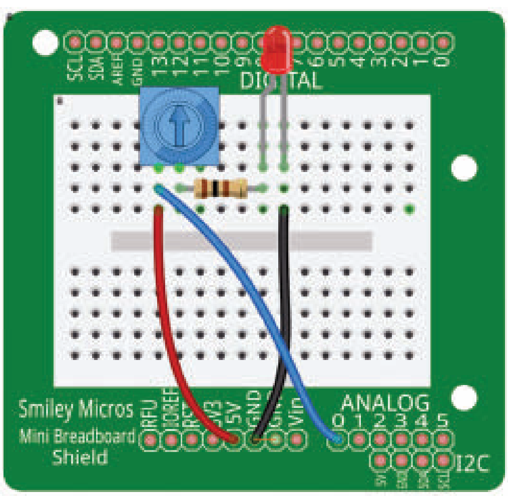


FIGURE 9: *Pot as rheostat drawing.*

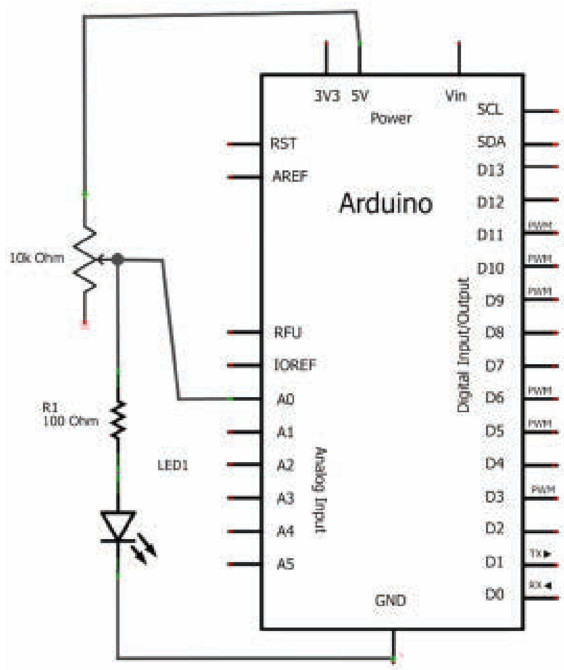


FIGURE 10: *Pot as rheostat schematic.*

```
value

void setup() {
  Serial.begin(57600);
  Serial.println("Measure potentiometer voltage
  rev 1.0");
}

void loop() {

  if(Serial.available())
  {
    char c = Serial.read();
    if(c == 'r')
    {
      // read the value from the sensor:
      sensorValue = analogRead(sensorPin);

      Serial.print("Potentiometer voltage: ");
      Serial.println(sensorValue);
      Serial.print("  Voltage: ");

Serial.println(((5.0*(float)sensorValue)/1024.0),
3);
    }
  }
}
```

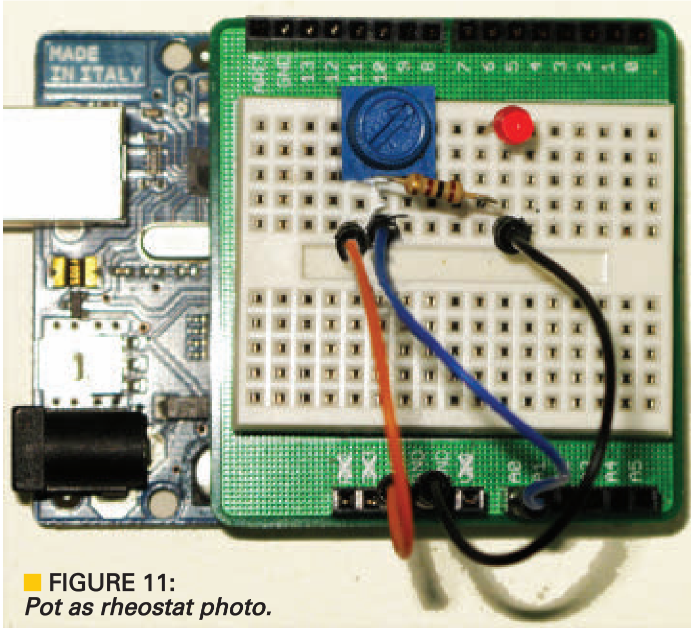


FIGURE 11:
Pot as rheostat photo.

❑ Run the program and the Arduino serial monitor.
❑ Observe the voltages reported for each of the angles suggested in Lab 1. Are these voltages close to what you calculated using Ohm's law?

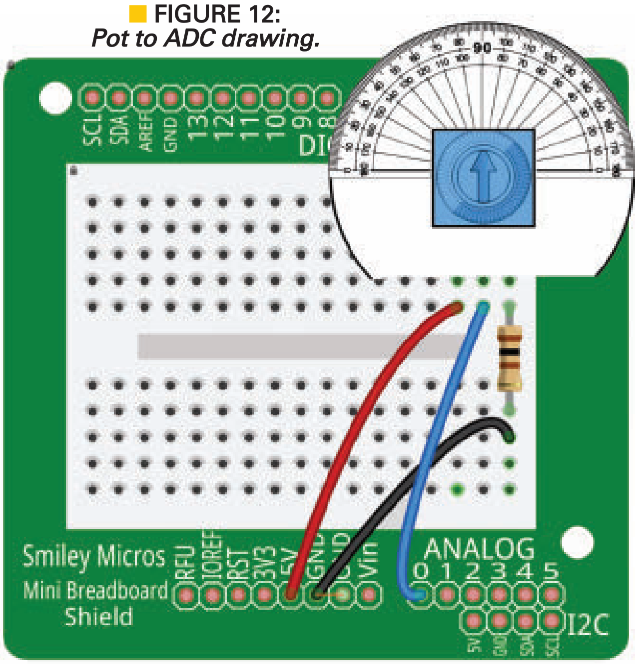
Lab 3: Dial — Using a Potentiometer to Input a Selected Angle on a Dial

Parts required:
1 Arduino
1 USB cable
1 Arduino proto shield and jumper wires
1 Potentiometer
1 100 Ω resistor
1 Potentiometer dial angle image

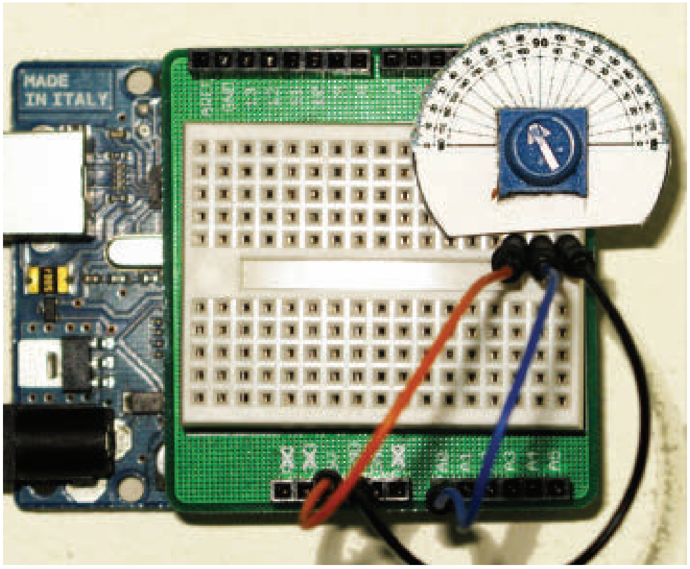
Estimated time for this lab: 30 minutes

Check off when complete:
❑ Print the pot dial image shown in **Figure 19** (available at the article link).
❑ Use double-sided sticky tape to stick the pot dial angle image to cardboard (cereal box thickness), then cut it out for use as shown in **Figures 12**, **13**, and **14**.
❑ Build the circuit shown in **Figures 12**, **13**, and **14**.
❑ Copy the following program into the Arduino IDE (again, available at the article link):

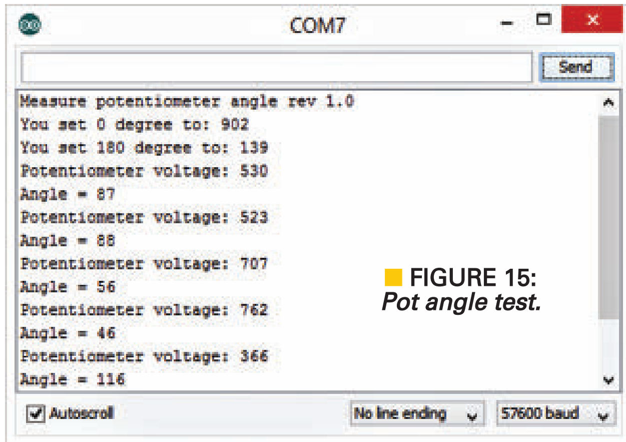
FIGURE 12:
Pot to ADC drawing.




```
// A101_ch7_pot_angle 5/7/14 Joe Pardue

#include <Servo.h>

Servo myservo;          // create servo object to
                        // control a servo

int sensorPin = A0;     // analog input pin
int sensorValue = 0;    // store the analog input
                        // value

int zero = 0;     // calibration reading for 0
                  // degree
int oneeighty = 0;   // calibrartion reading
                     // for 180 degree

void setup() {
  // attaches the servo on pin 9 to the servo
  myservo.attach(9);

  Serial.begin(57600);
  Serial.println("Measure potentiometer angle
  rev 1.0");
}
```

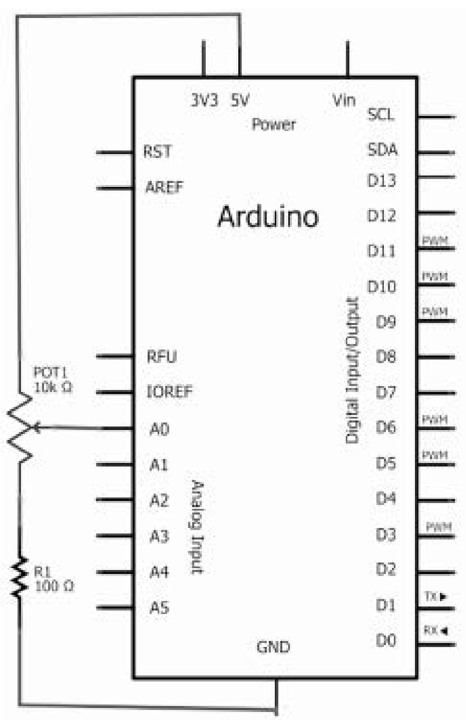


■ **FIGURE 14:** *Pot to ADC photo.*



■ **FIGURE 15:** *Pot angle test.*

```
void loop() {

  if(Serial.available())
  {
    char c = Serial.read();
    if(c == 'a') // get the zero degree
                 // calibration value
    {
      // read the value from the sensor:
      zero = analogRead(sensorPin);

      Serial.print("You set 0 degree to: ");
      Serial.println(zero);
    }
    if(c == 'b')      // get the 180 degree
                      // calibration value
    {
      // read the value from the sensor:
      oneeighty = analogRead(sensorPin);

      Serial.print("You set 180 degree to: ");
      Serial.println(oneeighty);
    }

  }
  delay(1000);
  int val = sensorValue;
  val = map(val,zero,oneeighty,0,180);

  // the dial is reversed for what we want so we
  // reverse the value to get the angle
  //val = 180 - val;
  myservo.write(val); // use converted angle
  Serial.println(val);
}
```



■ **FIGURE 13:** *Pot to ADC schematic.*

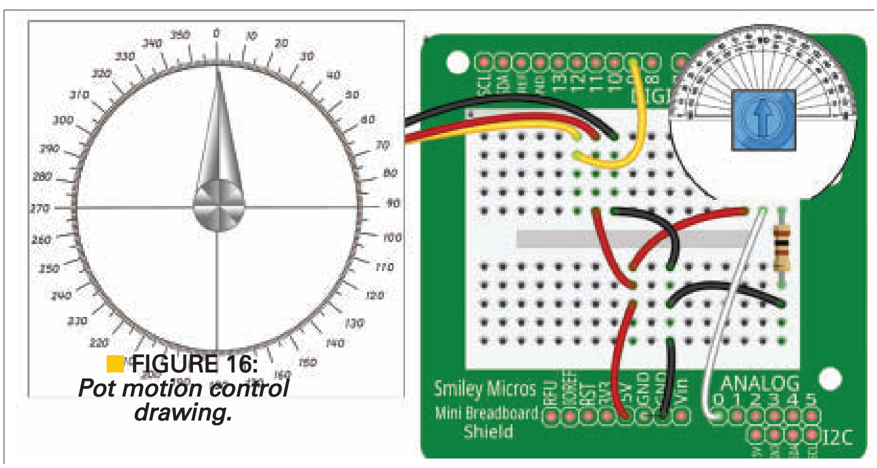
❑ This program calibrates the pot to the dial angles by accepting two commands: 'a' for the 0° position; and 'b' for the 180° position.

❑ Adjust the pot to point to 0°, then enter and send 'a' in the serial monitor.

❑ Adjust the pot to point to 180°, then enter and send 'b' in the serial monitor.

❑ Adjust the pot to point to 90°, then enter and send 'r' in the serial monitor to read the angle.

❑ Adjust the pot to various angles and read the angles with the serial monitor. Note the accuracy or lack of it for these readings. You should get results similar to those in **Figure 13**.



■ **FIGURE 16:** *Pot motion control drawing.*

Lab 4: Pot Motion Control

Now that we have a way to read an angle (not precisely but close), we will add the servomotor with the angle dial and pointer. We will then use the pot to control the position 10 times per second, and thereby control the motion of the servomotor.

Parts required:

1 Arduino
1 USB cable
1 Arduino proto shield and jumper wires
1 Potentiometer
1 100 Ω resistor
1 Potentiometer dial angle image
1 Servomotor
1 Servomotor angle dial image
1 Servomotor angle pointer image

Estimated time for this lab: 30 minutes

Check off when complete:

❑ The images for the pot dial, servomotor dial, and pointer are shown in **Figure 19**. You can reuse the servomotor dial and pointer from Chapter 5 if you have it. Construct the dial and pointer as discussed in Chapter 5.
❑ Assemble the servomotor dial and pointer, then plug it into the Arduino proto shield breadboard as in **Figures 1**, **16**, and **17**.
❑ Place the following program into the Arduino IDE:

```
// A101_ch7_pot_motion_control 5/7/14 Joe Pardue

#include <Servo.h>

Servo myservo;  // create servo object to control a
               // servo
int sensorPin = A0;  // analog input pin
int zero = 0; // calibration reading for 0 degree
int oneeighty = 0; // calibration reading for 180
                  // degree

void setup() {
  // attaches the servo on pin 9 to the servo
  myservo.attach(9);

  Serial.begin(57600);
  Serial.println("Pot motion control rev 1.0");
}

void loop() {
  int val;

  if(Serial.available())
  {
    char c = Serial.read();

    if(c == 'a') // get the zero degree calibration value
    {
      // read the value from the sensor:
      zero = analogRead(sensorPin);
      Serial.print("You set 0 degree to: ");
      Serial.println(zero);
    }
    if(c == 'b') // get the 180 degree calibration value
    {
      // read the value from the sensor:
      oneeighty = analogRead(sensorPin);
      Serial.print("You set 180 degree to: ");
      Serial.println(oneeighty);
    }
  }    delay(100);
```

Continued on page 77

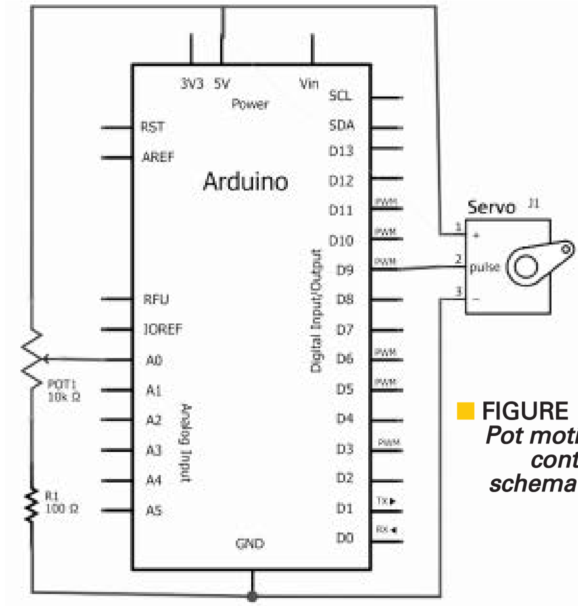


FIGURE 17: *Pot motion control schematic.*

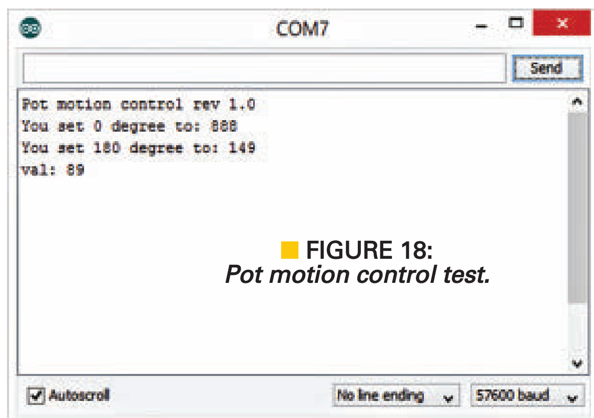
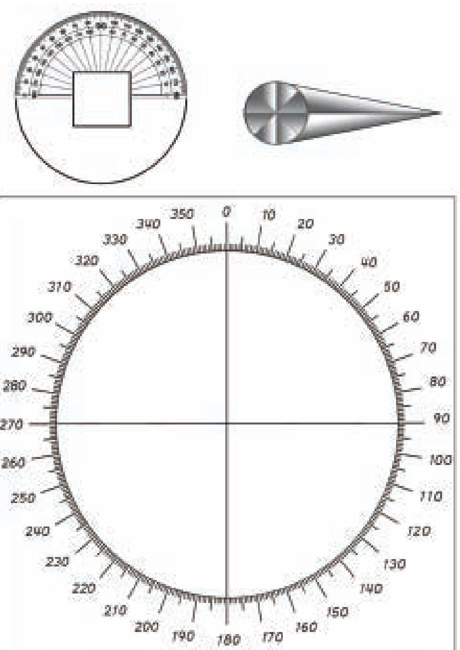


FIGURE 18: *Pot motion control test.*

FIGURE 19: *Servo and pot dials.*



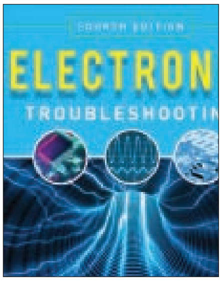
For complete product details, visit our webstore!!

The Nuts & Volts WEBSTORE

GREAT FOR DIYers!

NEW!

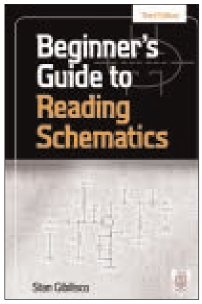
Electronic Troubleshooting, Fourth Edition
by Daniel Tomal, Aram Agajanian



Electronic Troubleshooting, Fourth Edition provides technicians with a wealth of problem-solving methods and information on troubleshooting theory, techniques, and practices for a wide variety of electrical and electronic devices. Special emphasis is placed on the digital electronics and microprocessor-based systems that are used in today's industrial and personal applications.

Regular Price $70.00
Sale Price $59.95

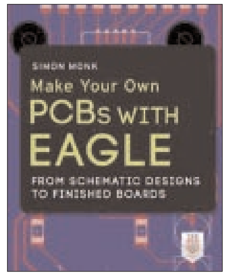
Beginner's Guide to Reading Schematics, 3E
by Stan Gibilisco



Navigate the roadmaps of simple electronic circuits and complex systems with help from an experienced engineer. With all-new art and demo circuits you can build, this hands-on, illustrated guide explains how to understand and create high-precision electronics diagrams. Find out how to identify parts and connections, decipher element ratings, and apply diagram-based information in your own projects.

$25.00

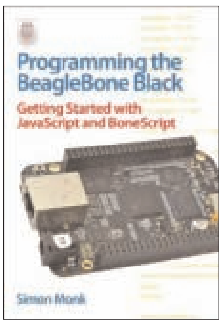
Make Your Own PCBs with EAGLE
by Eric Kleinert



Featuring detailed illustrations and step-by-step instructions, *Make Your Own PCBs with EAGLE* leads you through the process of designing a schematic and transforming it into a PCB layout. You'll then move on to fabrication via the generation of standard Gerber files for submission to a PCB manufacturing service. This practical guide offers an accessible, logical way to learn EAGLE and start producing PCBs as quickly as possible.

$30.00

Programming the BeagleBone Black: Getting Started with JavaScript and BoneScript
by Simon Monk

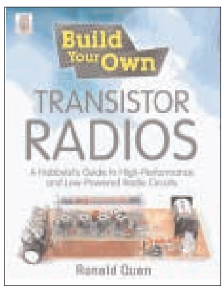


Learn how to program the BeagleBone Black — the wildly popular single-board computer — using JavaScript and the native BoneScript language. You'll find out how to interface with expansion capes to add capabilities to the basic board, and how to create a Web interface for BBB. Two hardware projects demonstrate how to use the board as an embedded platform.

$15.00

Build Your Own Transistor Radios
by Ronald Quan

A Hobbyist's Guide to High Performance and Low-Powered Radio Circuits

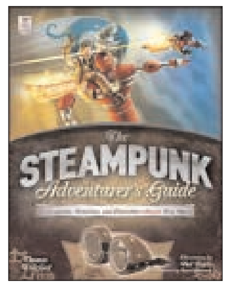


Create sophisticated transistor radios that are inexpensive yet highly efficient. Inside this book, it offers complete projects with detailed schematics and insights on how the radios were designed. Learn how to choose components, construct the different types of radios, and troubleshoot your work.

***Paperback, 496 pages**

$49.95

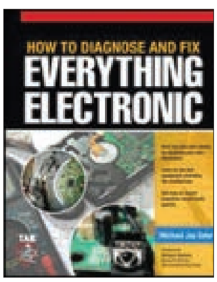
The Steampunk Adventurer's Guide
by Thomas Willeford



Steampunk stalwart Thomas Willeford cordially invites you on an adventure — one in which you get to build ingenious devices of your own! Lavishly illustrated by award-winning cartoonist Phil Foglio, *The Steampunk Adventurer's Guide: Contraptions, Creations, and Curiosities Anyone Can Make* presents 10 intriguing projects ideal for makers of all ages and skill levels, woven into an epic tale of mystery and pursuit.

$25.00

How to Diagnose and Fix Everything Electronic
by Michael Jay Geier



Master the Art of Electronics Repair

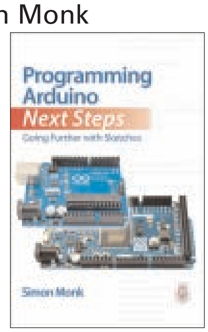
In this hands-on guide, a lifelong electronics repair guru shares his tested techniques and invaluable insights. *How to Diagnose and Fix Everything Electronic* shows you how to repair and extend the life of all kinds of solid-state devices, from modern digital gadgetry to cherished analog products of yesteryear.

$24.95

Programming PICs in Basic
by Chuck Hellebuyck

If you wanted to learn how to program microcontrollers, then you've found the right book! Microchip PIC microcontrollers are being designed into electronics throughout the world and none is more popular than the eight-pin version. Now the home hobbyist can create projects with these little microcontrollers using a low cost development tool called the CHIPAXE system and the Basic software language. Chuck Hellebuyck introduces how to use this development setup to build useful projects with an eight-pin PIC12F683 microcontroller. **$14.95**

Programming Arduino Next Steps: Going Further with Sketches
by Simon Monk



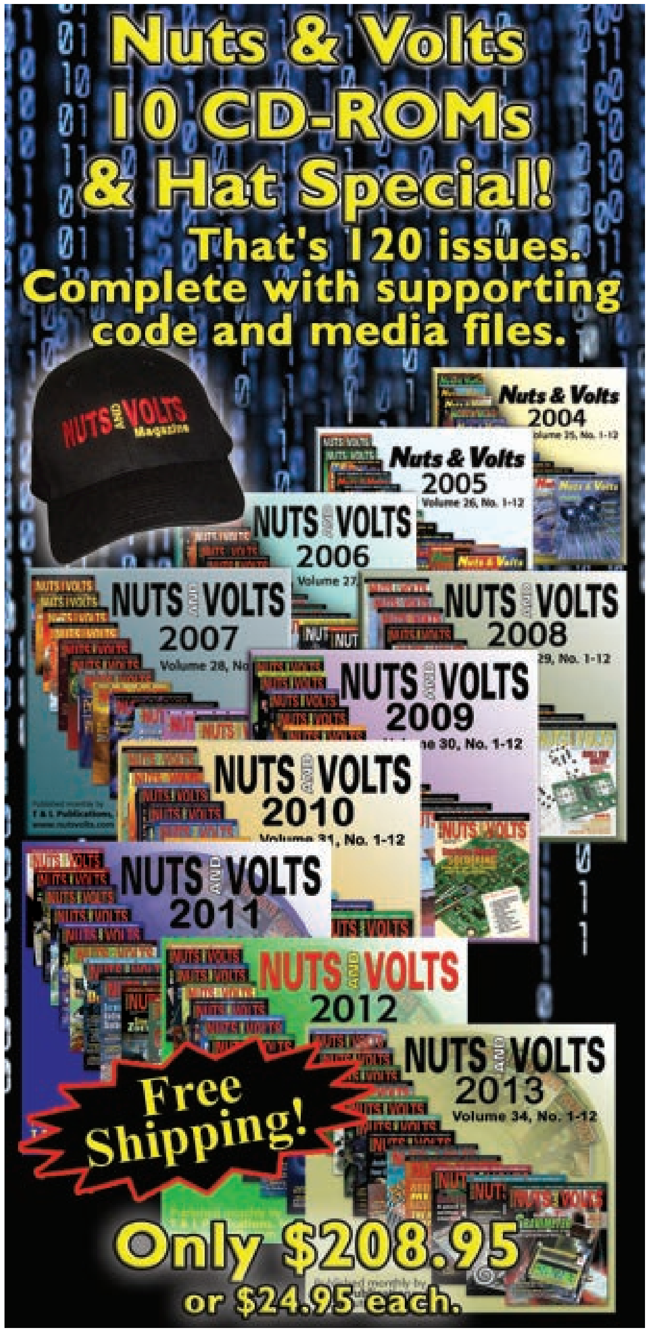
In this practical guide, electronics guru Simon Monk takes you under the hood of Arduino and reveals professional programming secrets. Also shows you how to use interrupts, manage memory, program for the Internet, maximize serial communications, perform digital signal processing, and much more. All of the 75+ example sketches featured in the book are available for download. **$20.00**

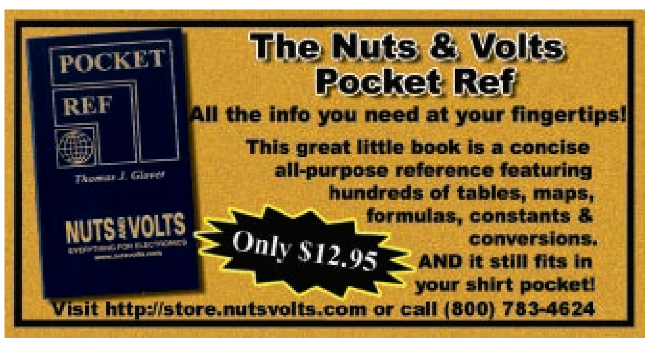
Order online @ www.store.nutsvolts.com

Or CALL 1-800-783-4624 today!

BOOK & KIT COMBOS

CD-ROM SPECIAL

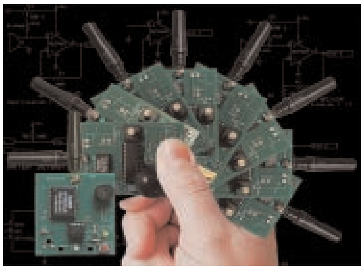




CALL 1-800-783-4624 today!
Order online @ www.nutsvolts.com

PROJECTS

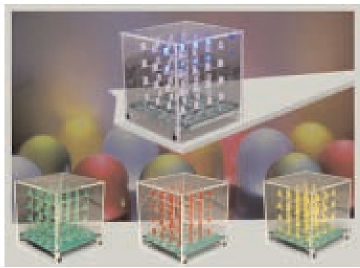
Super Detector Circuit Set



Pick a circuit!
With one PCB you have the option of detecting wirelessly: temperature, vibration, light, sound, motion, normally open switch, normally closed switch, any varying resistor input, voltage input, mA input, and tilt, just to name a few.

$32.95

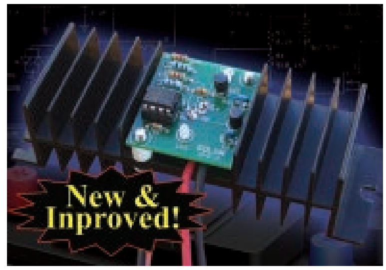
3D LED Cube Kit



This kit shows you how to build a really cool 3D cube with a 4 x 4 x 4 monochromatic LED matrix which has a total of 64 LEDs. The preprogrammed microcontroller that includes 29 patterns that will automatically play with a runtime of approximately 6-1/2 minutes.
Colors available: Green, Red, Yellow & Blue

$57.95

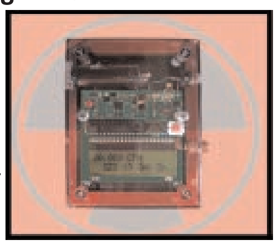
Solar Charge Controller Kit 2.0



Now with more options!
If you charge batteries using solar panels, then you can't afford not to have them protected from over-charging. This 12 volt/12 amp charge controller is great protection for the money. It is simple to build, ideal for the novice, and no special tools are needed other than a soldering iron and a 9/64" drill!

$27.95

Geiger Counter Kit



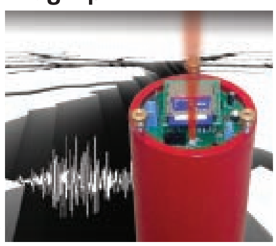
As seen in the March 2013 issue.

This kit is a great project for high school and university students. The unit detects and displays levels of radiation, and can detect and display dosage levels as low as one micro-roentgen/hr. The LND712 tube in our kit is capable of measuring alpha, beta, and gamma particles.

Partial kits also available.

$145.95

Seismograph Kit



As seen in the May 2012 issue.
Now you can record your own shaking, rattling, and rolling.
The Poor Man's Seismograph is a great project /device to record any movement in an area where you normally shouldn't have any. The kit includes everything needed to build the seismograph. All you need is your PC, SD card, and to download the free software to view the seismic event graph.

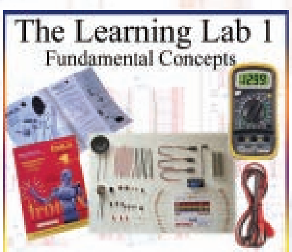
$79.95

Battery Marvel

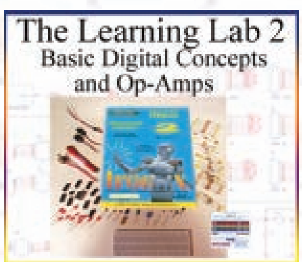
As seen in the November 2011 issue.
Battery Marvel helps protect cars, trucks, motorcycles, boats, and any other 12V vehicles from sudden battery failure. This assembled unit features a single LED that glows green, yellow, or red, indicating battery health at a glance. An extra-loud piezo driver alerts you to any problems.
Details, please visit our website.

$32.95

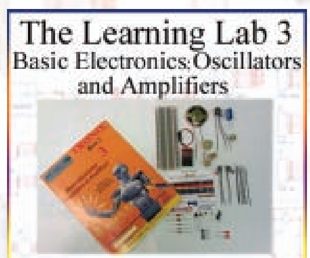
FOR BEGINNER GEEKS!



$59.95



$49.95



$39.95

The labs in this series — from GSS Tech Ed — show simple and interesting experiments and lessons, all done on a solderless circuit board.

As you do each experiment, you learn how basic components work in a circuit, and continue to build your arsenal of knowledge with each successive experiment.

For more info and a promotional video, please visit our webstore.

CLASSIFIEDS

SMILEY'S WORKSHOP

Continued from page 73

```
   val = analogRead(sensorPin);;
   val = map(val,zero,oneeighty,0,180);

   // the dial is reversed for what we want so we
   // reverse the value to get the angle
   val = 180 - val;
   myservo.write(val); // use converted angle
}
```

❑ Run the program and open the serial monitor. Set the left and right calibration points.
❑ Set the pot angle to 90° and input 'r' to read the angle as shown in **Figure 18**.

See you next time! **NV**

READER-TO-READER TECHFORUM

>>> QUESTIONS

Photoresister Switcher

I would like to know if I have correctly connected the photoresistors (CdS photocells) shown here in order to turn OFF during the day and to turn ON during the night the two LED circuits attached to them. (If not, please indicate by a new diagram.)

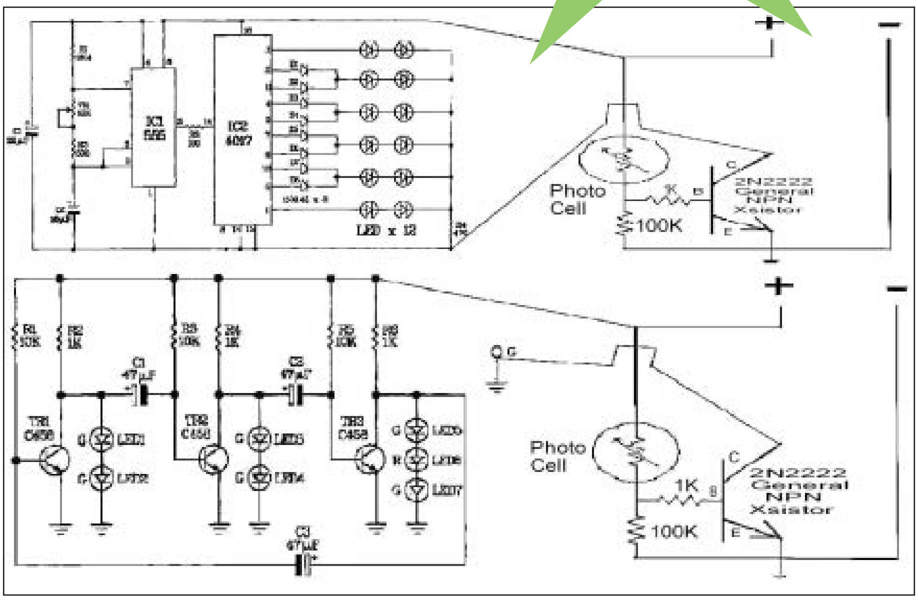
Also, I would like to know:

a) If any photoresistor would work?

b) What would the optimum dark/light resistance values be for such a photoresistor?

c) How would I calculate the values (any formula?) from the transistor side (2N2222) that would best fit this ON/OFF photoresistor switcher?

#7141 **Nate**
via email



Wireless Mirrors

I have an old car that came with electric windows but has old fashioned side view mirrors which are almost useless. I would like to upgrade to more modern mirrors that are aimed by control of 12V electric motors. On the driver's side, that is no problem as you simply use a four-way switch scavenged off the vehicle that you got the mirrors from, using the 12V available to run the electric windows.

However, to use that same switch to control the right hand mirror, I would have to drill a hole in the edge of both doors and both front pillars, and snake a wire through from the left side to the right side. It occurred to me that the use of radio waves from the left hand door could be used to control four relays in the right hand door and obtain the required action.

Something like Ron Newton used in his "Super Detector" article in the October 2013 issue of *NV*. Any suggestions would be appreciated.

#7142 **Dean Kaul**
Kalamazoo, MI

Lithium Charging

I'm thinking of building a solar charger for my iPhone, but don't know how to handle the internal lithium battery in terms of taper current, etc. — especially when I have the phone on all day. What I've found online is information on charging disconnected lithium batteries, not ones under load. Any hints?

#7143 **Anthony Suchek**
White Plains, NY

>>> ANSWERS

#4144 - April 2014]
Receiver ... Not

I inherited an old tube-type short wave receiver from my grandfather. It produces nothing but static, even after I replaced all of the vacuum tubes. All of the switches are good.

#1 Nearly any serious restorer of antique electronics — especially old radios — will first replace all electrolytic and paper dielectric capacitors in the receiver. This is especially true if the radio has been in storage for a long time. Electrolytics dry out and power supply filtering and decoupling become poor. Paper dielectric caps get leaky over time — but paper only. Don't worry about ceramic and mica caps at this point,

Send all questions and answers by email to **forum@nutsvolts.com**
or via the online form at **www.nutsvolts.com/tech-forum**

and don't play with any caps in the "front end" of the receiver as these are rarely paper except for a few that may be in the power supply decoupling circuits. These, you replace.

A leaky coupling cap on the grid of an audio preamp in a higher power output amp can cause that tube to go into full conduction and drive one of the output tubes full-on to the point that it will burn out the primary of the output transformer — a common problem in many old console radios.

Your best friends will be found on the forum at **www.antiqueradios.com**. These guys — if they don't own your particular model — will know about it, and will be able to help you tremendously in its repair and restoration. Don't visit the forum as a lurker. Sign up. It's free (but they'd like a donation, if possible). Then, you can be set to be notified when there's activity on your thread.

Also, don't throw away all those tubes you replaced. There's probably nothing wrong with them.

Dean Huster
via email

#2 Since you are producing "static," you know the power supply, final amplifiers, and speakers are okay. Short of having a schematic and using a signal generator and oscilloscope to check for proper operation, I would replace all of the capacitors. Old capacitors have a tendency to go bad more so than the inductors. An old technique involved "jumpering" a known good capacitor around a suspect one, but in the case of an old unit, replacement may be the best approach.

Also check for cracked resistors and loose wiring (I am assuming this unit pre-dates circuit boards), and clean the switches, contactors, and tuning capacitors. IF this fails, you need a signal generator (frequencies covering RF and IF for the unit) and an oscilloscope to check the circuit for the faulty stage(s).

Tim Brown PhD EE, PE
Honea Path, SC

#3 We need more information about the receiver. What bands (frequency range) does it cover and what antenna are you using? Generally, you need a long wire type antenna that is 100' or longer mounted as high as you can get it. Propagation will affect what stations you can hear. You may be listening to a "dead" band that will come to life at another time of day.

Also, many shortwave stations have been relocated to different frequencies in the last 10 years to free up spectrum for other use. It's possible your radio is listening in the spectrum where stations have vacated. Finally, the restoration of a "boat anchor" is often more complex than just a re-tube and clean-up can fix. A common failure is with old electrolytic and paper capacitors.

Perry Ogletree
Murfreesboro, TN

[#5144 - May 2014]
What Wall Wart?

While on vacation, I managed to lose the power supply wall wart to a cheap vintage portable no-name brand shortwave radio. On the back of the radio, the power jack says 9 VDC, and has a C shaped circle and a dot in the center with a plus sign on the dot. I'm pretty sure this means nine volts; positive tip. However, what it doesn't state is the milliamp rating. If I use a power supply that has too high or low an amperage rating, am I in danger of damaging the radio?

#1 The symbol you see on the radio indicates a "coaxial" power adapter plug with "tip positive" polarity. Go to your local RadioShack and get a "Universal" wall wart that will deliver 9 VDC at 1,500 (or more) mA DC. (**IMPORTANT:** Take your radio to the store so you can get the proper "adaptaplug" to use with the wall wart!)

I suggest a 1,500 mA (i.e., 1.5A) or larger current rating to ensure you'll have enough power. DO NOT FEAR: A *wart* with a *large* current rating WILL NOT harm the radio. However, a *smaller-than-needed* current rating will quickly burn out the wall wart (or cause overheating and/or fire hazard).

Hope this helps, and happy listening to your radio.

Ken Simmons
Auburn, WA

#2 When choosing a wall wart, you have to check three things and look out for a problem with the wall warts.

1. Does the voltage match?
2. Does the polarity match?
3. Does the wall wart put out enough current?

The catch is with the voltage. Wall warts come in four different flavors:

1. Transformer — unregulated.
2. Transformer — regulated.
3. Switching — unregulated (very rare).
4. Switching — regulated.

The belief that the general public has about power supplies is that the supply will drive whatever current it is rated at.

This is WRONG in capital letters. The device will draw whatever current it is rated at for the rated voltage. The power supply must be able to supply at least as much current as the device needs. If it can't, the voltage will fall off. If it is rated at more current than necessary, this won't hurt anything as long as the maximum voltage is not exceeded. This is the catch.

This is why it's important to check *unregulated* wall warts at their rated current. A nine volt unregulated power supply may read as high as 18 volts with no load on it. So, if an unregulated supply is rated at one amp at nine volts and the device only needs 250 mA or 1/4, amp the voltage output of

the supply could be 12, 14, or even 18 volts. *OOPS, here comes the smoke!*

A regulated wall wart will put out the same voltage plus or minus some percentage loaded or unloaded. A switching wall wart will be very light compared to a transformer wall wart of the same amperage rating.

Yes, the tip needs to be positive.

Richard Pope
via email

[#6142 - June 2014]
Delay Circuit Needed

I need a little help with an automatic above-ground pool filler circuit I constructed. It's nothing complicated — a water level switch with a power supply. When the water level drops below the set point of the float switch, it completes the circuit which applies 12 VDC to a solenoid which activates and allows the water to flow.

When the water level comes up and the float switch raises up to the preset point, the reverse happens and the water stops flowing.

My problem is I need some sort of delay before the power supply sends 12 VDC to the water relay. The float switch — due to motion by the wind or vibration — normally keeps the water level switch bouncing. This, in turn, keeps the water constantly turning on and off. If there was a 15 to 30 second delay until the float switch stabilized in one condition — either on or off — it would prevent the water switch from constantly going on and off.

Anyone out there with a circuit that can do this? It would be even better if the same timer also supplied the voltage for the water relay.

#1 An effective method to prevent short-term fluctuations from exercising the float switch is to dampen the frequent rise and fall of the water level by a non-electronic method. Just enclose the float switch in a vessel (such as a one gallon milk container with the top cut off) and make a very small hole in the bottom of the container. This is the equivalent of adding a large capacitor to a varying DC voltage to smooth out the variations. The float switch will respond to an average level over time. Depending on the size of hole, it will take some time for the water level in the container to rise or fall. I believe this is essentially what lake level monitors use to tune out temporary wave action.

Bryan Carl
Marietta, GA

#2 From your description, it looks like the wave motion is exceeding the movement differential or differential gap of your switch. I have a couple of solutions.

1. *Mechanical Solution (easy):* Insert the float in a piece of PVC pipe which is large enough to allow free movement of the float. In industrial applications, this is called a stilling well and it works by shielding the float from the wave action of the pool surface.

2. *Electrical Solution (more complicated):* By using two Normally Closed (NC) float operated switches and a 12V relay with an extra set of NO contacts, you can build a circuit to turn the solenoid valve on at a "low" level and off at a "high" level. Wire the relay coil, low float switch, and high float switch in series between the +12V source and ground. Wire the "extra" relay contact in parallel with the low float switch and the other relay contact in series with the solenoid valve. This arrangement works as follows:

(a) The two float switches are open when the float is high; when the level drops to the low level, the low float switch closes and activates the relay which closes the relay contacts.

(b) As water runs into the pool, the low float switch opens up but the "hold in" contact on the relay keeps the relay (and thus the solenoid valve) activated, allowing the pool to continue filling.

(c) When the pool level reaches the high float, the high float switch activates (opens), and the relay and solenoid valve deactivate stopping the flow of water. You can experiment with the low level position to obtain the operation you want.

CAUTION: *Water and electricity do not mix. Any electrical devices used around water where humans can contact the electrified water should be fed through a ground fault interrupt device.*

Tim Brown PhD EE, PE
Honea Path, SC

#3 The easiest solution is to use a time delay relay. If the relay is on for 30 seconds straight, then the contact will close for the relay. So, wire the float switch in series with the time delay relay coil. Then, wire power through the relay contact to the water valve. The water valve will come on when the float is low for 30 continuous seconds — until the float switch opens up.

There's a 30 sec and a 60 sec time delay relay with socket available at **www.mpja.com** for $9.95. You should probly put a reverse diode across the time delay relay so you don't burn out the contacts on the water float switch. If you need a schematic, email me at jims@ruralhost.com and I should be able to come up with one for you.

Schneids
Redgranite, WI

[#6143 - June 2014]
Battery Charging Indicator

My truck only has a voltmeter for battery condition indication. I don't have room for an ammeter. Is there a circuit I could build with a red and green LED to indicate if the battery is charging or discharging?

#1 If you want to know if the charging circuit is working, note the voltage with the switch on, motor off; then start the motor. The voltage should go up about one volt. If it goes into the red, the regulator is defective.

Russ Kincaid
Milford, NH

#2 I suspect that your truck already

has the circuit installed. If it is less than 20 years old, it almost certainly does. It is called the "battery indicator light." That light does exactly what you want, except it is dark when you want an illuminated green LED.

Put the keys in the ignition and switch to the ON position without starting the engine. All the warning lights on the dash should light up (this is a bulb check routine to identify burned out indicator bulbs or LEDs). After a couple seconds, most of the warning lights will go off, except probably two. One is the oil pressure warning light. The engine isn't running, so the oil pressure is zero, so the light is on. The other light still on will be the battery light. The engine isn't running so the battery is supplying all the power to the truck's electrics.

Now, start the engine. Those two lights should go off. The battery light is off which means the alternator is supplying all the power to the truck's electrical systems. That little indicator light basically just looks at the electrical system's voltage. A good, fully charged car lead-acid battery will read 12.6V without any load. The charging system of a car runs at about 13.5V to 14V.

You can check the voltages with a voltmeter at the battery posts with the engine on and off. If the electrical system is at a voltage of 12.6V or less, then the battery is carrying the electrical load. If the electrical system is at more than about 13V, then the alternator is supplying all the electrical power. If the battery needs charging, then the alternator is doing so — as long as the battery indicator light is off. If the battery is not charging because it doesn't need to be, then the light is also off. What the battery light won't tell is the charge/discharge rate which you would need an ammeter for.

Jim Sluka
Greenwood, IN

■ LOOK FOR ■ SEARCH FOR ■ FIND

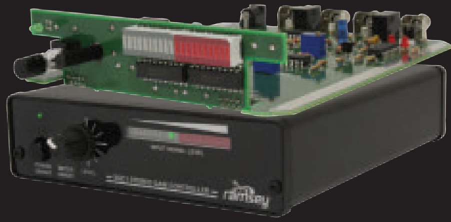
Find your favorite advertisers here!

ADvertiser INDEX

Beat The Heat...Build a Kit!

Ramsey Kits Are Always Cool, Even In The Summer Heat!

Stereo Audio Platform Gain Controller



- ✔ Stereo audio processing while preserving audio dynamics!
- ✔ True stereo control keeps virtual sonic source location intact!
- ✔ Automatic set-and-forget operation!
- ✔ Auto-bypass restores original levels when power is turned off!
- ✔ Unbalanced RCA and 3.5mm stereo input and output line level jacks!
- ✔ Built-in bar graph indication of signal level with display mute!

Your Full-Time Audio Engineer... Without the Payroll!

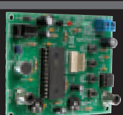
The SGC1 is one of our latest kits, and provides a great solution to the age-old problem: how can we easily correct inconsistent audio levels without negatively affecting the dynamics of the audio signal? The SGC1 circuit implements a principle known as the "Platform Gain Principle," which was originally developed by CBS Labs (what we now know as CBS in the TV and radio world) to allow transmitted audio levels to be automatically adjusted to keep them within a desired range.

Think of it like an audio engineer, constantly adjusting the output level in order to limit highs that would be too loud while boosting lower levels so that they can still be heard. You may think "oh, this is just another limiter/compressor!" Not so! Here's the real trick: keeping the full dynamic range ratio of the output signal the same as the original input - something the typical limiter/compressor can only dream of doing! The SGC1 can be placed in just about any standard analog stereo line level audio circuit (the red and white RCA connectors or the mini-phone connector) to keep the audio level within the desired range. It's also the perfect addition to any of our hobby kit transmitters, allowing you to match levels between different audio sources while keeping lows audible and preventing the highs from overdriving.

The SGC1 makes a great addition to any audio system where you need to keep levels from different sources under control, but still make sure they all sound great! In addition to its useful basic function and great audio performance, the SGC1 also boasts a front panel LED meter to give an indication of the relative level of the input signal, plus a level control (also on the front panel) that allows you to adjust the controller to the min/max center point of your desired level range. And yes, it is a **S**tereo **G**ain **C**ontroller! Meaning that the levels of both the left and right channels are monitored and adjusted equally, thereby maintaining the relative virtual position of things like instruments, singers and speakers! When powered down the controller is automatically bypassed, and will simply pass through the unchanged original audio, meaning no messing around with cables required if you simply want to cut it out of the equation temporarily. There's even a front panel switch to disable the LED display - perfect for situations where light can be distracting - like a darkened home-theatre! The entire unit is housed in a slim attractive black textured aluminum case that is sure to complement your studio or home theatre. If you're looking for perfect audio levels, hire a broadcast audio engineer, but if that doesn't fit your budget, the SGC1 is the next best thing! Includes 15VDC world-wide power adapter.

SGC1 Stereo Audio Platform Gain Controller Kit, with Case and Power Supply $179.95

Audio Recorder & Player



Record and playback up to 8 minutes of messages from this little board! Built-in condenser mic plus line input, line & speaker outputs. Adjustable sample rate for recording quality. 4-switch operation that can be remote controlled! Runs on 9-12VDC at 500mA.

K8094 Audio Recorder/Player Kit $32.95

Water Sensor Alarm

This little $7 kit really shined during Sandy! Simply mount the alarm where you want to detect water level problems. When the water level rises it touches the contacts and the alarm goes off! Sensor can even be remotely located. Runs on a 9V battery (not included).

MK108 Water Sensor Alarm Kit $6.95

12VDC Regulated

Go green with our new 12VDC 1A regulated supply. Worldwide input 100-240VAC with a Level-V efficiency! It gets even better, includes DUAL ferrite cores

AC121 12VDC 1A Regulated Supply $9.95

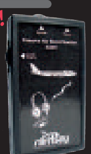
Electronic Watch Dog

A barking dog on a PC board! And you don't have to feed it! Generates 2 different selectable barking dog sounds. Plus a built-in mic senses noise and can be set to bark when it hears it! Adjustable sensitivity... unlike my Greyhound and Boxer! 9-12VDC.

K2655 Electronic Watch Dog Kit $39.95

Passive Aircraft Monitor

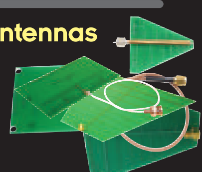
PATENTED!



The hit of the decade! Our patented receiver hears the entire aircraft band without any tuning! Passive design has no LO, therefore can be used on board aircraft! Perfect for airshows, hears the active traffic as it happens! Available kit or factory assembled.

ABM1 Passive Aircraft Receiver Kit $89.95

Precision PC Plane Antennas



Our LPY series PC antennas continue to be the favorite for virtually all RF and wireless applications. Frequencies:

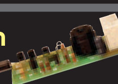
- 400-1000 MHz Log Periodic
- 2.1-11.0 GHz Log Periodic
- 2.4 GHz 8dB Yagi
- 1250-1300 MHz Yagi
- 2.4 GHz 12 dB Patch
- 900-2600 MHz Log Periodic
- 915 MHz Yagi

Complete with cable & connector... ready to use!

From microwave links, wireless mics, 802.11 WAPs, to RFID, we've got you covered. Check our site for complete specifications and details!

LPYSeries Precision PC Plane Antennas from $29.95

Laser Trip Sensor Alarm



True laser protects over 500 yards! At last within the reach of the hobbyist, this neat kit uses a standard laser pointer (included) to provide both audible and visual alert of a broken path. 5A relay makes it simple to interface! Breakaway board to separate sections.

LTS1 Laser Trip Sensor Alarm Kit $29.95

The Learning Center!

Beginners To Advanced... *It's Fun!*

- ✔ Learn, build, and enjoy!
- ✔ 130, 200, 300, & 500 in one electronic labs!
- ✔ Practical through hole and SMT soldering labs!
- ✔ Integrated circuit AM/FM radio lab!
- ✔ AM, AM/FM, and SWL radio labs!
- ✔ Radio Controlled (RC) car!
- ✔ Beginner's non-soldering kits!

For over 4 decades we've become famous for making electronics fun, while at the same time making it a great learning experience. As technology has changed over these years, we have continued that goal!

PL130A Gives you 130 different electronic projects together with a comprehensive learning manual describing the theory behind all the projects.

PL200 Includes 200 very creative fun projects and includes a neat interactive front panel with 2 controls, speaker, LED display and a meter.

PL300 Jump up to 300 separate projects that start walking you through the learning phase of digital electronics.

PL500 The ultimate electronics lab that includes 500 separate projects that cover it all, from the basics all the way to digital programming.

SP1A Whether young or old, there's always a need to hone your soldering skills. Either learn from scratch or consider it a refresher, and end up with a neat little project when you're done!

SM200K Move up to Surface Mount Technology (SMT) soldering, and learn exactly how to solder those tiny little components to a board!

AMFM108K We not only take you through AM and FM radio theory but we guide you through IC's. When you're done you've built yourself an IC based AM/FM radio that works great!

AM2 Learn the complete theory of AM broadcast radio and end up with a highly sensitive AM radio receiver!

AMFM7 Step up to AM/FM with this multi-band lab and learn the basics of both bands. The FM tuner is factory assembled and aligned!

SR3 Enter the world of SWL with this shortwave radio learning lab covering 6-8MHz and 12-18MHz!

AK870 One of the most exciting electronic learning kits that the kids will love! Build a complete RC speedster from the ground up! 7 remote functions!

PL130A	**130-In-One Lab Kit**	**$39.95**
PL200	**200-In-One Lab Kit**	**$84.95**
PL300	**300-In-One Lab Kit**	**$109.95**
PL500	**500-In-One Lab Kit**	**$249.95**
SP1A	**Through Hole Soldering Lab**	**$9.95**
SM200K	**SMT Practical Soldering Lab**	**$22.95**
AMFM108K	**AM/FM IC Lab Kit & Course**	**$36.95**
AM2	**AM Radio Learning Lab**	**$11.95**
AMFM7	**AM/FM Radio Learning Lab**	**$12.95**
SR3	**Short Wave Radio Learning Lab**	**$16.95**
AK870	**RC Speedster Car Kit**	**$29.95**

ramsey
www.ramseykits.com

There's only so much room on these two pages, so check it all out in our new virtual electronic catalog! Flip through the pages and search with ease! Visit ***www.ramseycatalog.com***

Follow Us and SAVE $$

Follow us on your favorite network site and look for a lot of super deals posted frequently... exclusively for our followers!

- ✔ **Our latest generation Nixie clock, 6 or 4 tube!**
- ✔ **Today's latest technology with yesterday's display!**
- ✔ **Choice of our signature hand rubbed Teak/Maple or acrylic base!**
- ✔ **Choice of Nixie tubes types (IN-14 or IN-82)**
- ✔ **Low-cost GPS receiver timebase option!**
- ✔ **Programmable color LED tube mood lighting!**
- ✔ **The ultimate conversation piece!**

THE NEW STANDARD IN NIXIE CLOCKS!

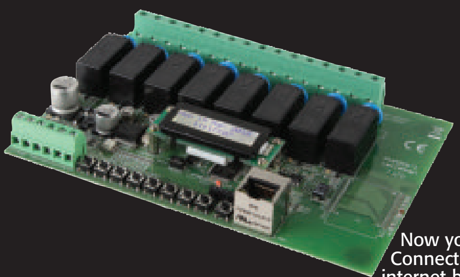
Our next generation of classic Nixie tube clocks perfectly mesh today's technology with the Nixie era technology of the 60's. Of course, features you'd expect with a typical clock are all supported with the Nixie clock... and a whole lot more! Time wise, the clocks are designed around a quartz crystal timebase, therefore are not AC power frequency dependent like a lot of clocks. This means they can be used in any country regardless of power frequency, with the included 12VDC regulated power supply. The clocks are also programmable for 12 or 24 hour mode, various AM/PM indications, programmable leading zero blanking, and include a programmable alarm with snooze.

Unlike most Nixie clocks, the clocks also display the date in DD.MM.YY, MM.DD.YY, or YY.MM.DD format, which can be programmed to display for a few seconds at the end of each minute either as a static display, or by a neat scrolling in/out from alternating sides of the display. Display wise, the clocks feature a programmable night mode with dim or blank display, a programmable master blank tube saver, hard or soft fade digit change, and even have a built-in "Slot Machine" cathode poisoning prevention routine. Programming and setting the clock is a breeze with simple 2-button entries on the rear panel. The clocks are available in our signature hand rubbed Teak & Maple or futuristic clear acrylic bases.

WE CROSSED THE TECHNOLOGY TIMELINE!

We then jumped the technological timeline of the 60's Nixie displays by adding the latest multi-colored LEDs to the base of the Nixie tubes to provide hundreds of illumination colors to highlight the glass tubes! The LED lighting can be programmed to any color and brightness combination of the colors red, green, or blue to suit your mood or environment. Then we leaped over the technological timeline by integrating an optional GPS time base reference for the ultimate in clock accuracy! The small optional GPS receiver module is factory assembled and tested, and plugs directly into the back of the clock to give your Nixie clock accuracy you could only dream of! The new series clocks are available in 6-tube and 4-tube versions, with your choice of bases, and your choice of kit or factory assembled & tested. If you're looking for the ultimate conversation piece, with a trip down nostalgia lane, check out our clocks at www.ramseykits.com/nixie.

NIXIE CLOCKS	**Classic Nixie Tube Clocks, Teak/Maple or See-Through Acrylic Base, Kit or Factory Assembled**	**From $229.95**



Remotely Control and Monitor Your Equipment On-Line!

8-Channel Ethernet Remote Controller

- ✔ **8 separate relay controls!**
- ✔ **Built-in web interface for easy operation!**
- ✔ **User defined TCP server port!**
- ✔ **Static or DHCP IP address!**
- ✔ **Sends email at power-up and event changes!**
- ✔ **Multiple user authentications**
- ✔ **On-board LCD display**

Now you can easily control and monitor up to 8 separate circuits via the standard Ethernet network in your home or office. Connection wise it couldn't be simpler. The controller functions as an IP based web server, so it can be controlled by any internet browser that can reach your network! There are no drivers or proprietary software required, just access the controller like any web page from your PC, laptop, or even your smartphone! Security is assured allowing up to 4 separate user credentials. The controller can be set to a specific static IP within your network subnet or can be set to DHCP (auto negotiate). The controller can even be programmed to send you an email to notify and confirm power up and status changes!

To simplify the connection of your equipment to the controller, 8 separate and isolated relay outputs are provided! This gives you internet or network control of up to 8 separate functions. No need to worry about interfacing a logic high or logic low, or burning up the interface! The applications are endless! From something as simple as turning on and monitoring lights at your house with a normal latched closure to advanced control of your electronic gadgets, radio equipment, or even your garage door! Each relay contact is rated at 12A at 30VDC or 16A at 230VAC. Each of the 8 channels has built-in timer and scheduler programs for day, weekend, working days, every day, and every day except Sunday. Relay control functions are programmable for on, off, or pulse (1-99 seconds, 1-99 minutes, or 1-99 hours). In addition to control functions, the web interface also displays and confirms the status of each channel. Each channel can be custom labeled to your specific function name. The controller operates on 12VDC or 12VAC at 500mA or our new AC121 global 12VDC switching power supply over to the left! Factory assembled, tested, and ready to go! Even includes a Cat-5 cable!

SPECIFICATIONS

Channels:	8, with separate relay outputs
Format:	IEEE 802.3 compatible Ethernet, 10Mbps
TCP/IP:	IPV4, static or DHCP
NETBios:	User defined name
Server:	TCP, SNTP, email
TCP Port:	User defined
Display:	On-board LCD, time or network status
Labels:	User defined for input and all outputs
Users:	Administrator plus 3 users
Timers:	28 times max
Contacts:	12A@30VDC, 16A@230VAC x 8
Power:	12VDC, 12VAC, 500mA
Mounting:	PC board or optional DIN rail enclosure

VM201	**8-Channel Remote Ethernet Controller, Factory Assembled & Tested**	**$169.95**

GET THE NUTS AND VOLTS DISCOUNT!

*Mention or enter the coupon code **NVRMZ142** and receive 10% off your order!*

800-446-2295

www.ramseykits.com

Prices, availability, and specifications are subject to change. We are not responsible for typos, stupids, printer's bleed, or 4th of July fireworks accidents! In the true holiday spirit, Robin made me do this ad copy instead of going on the family picnic! Not Fair! Visit www.ramseykits.com for the latest pricing, specials, terms and conditions. Copyright 2014 Ramsey Electronics®...so there!

RAMSEY ELECTRONICS®
590 Fishers Station Drive
Victor, NY 14564
(800) 446-2295
(585) 924-4560